

ELEC-H-473

Microprocessor architecture

Intel x86 micro-architecture

Intro and NetBurst architecture

Lecture 09

Dragomir MILOJEVIC

dragomir.milojevic@ulb.ac.be

Previously ...

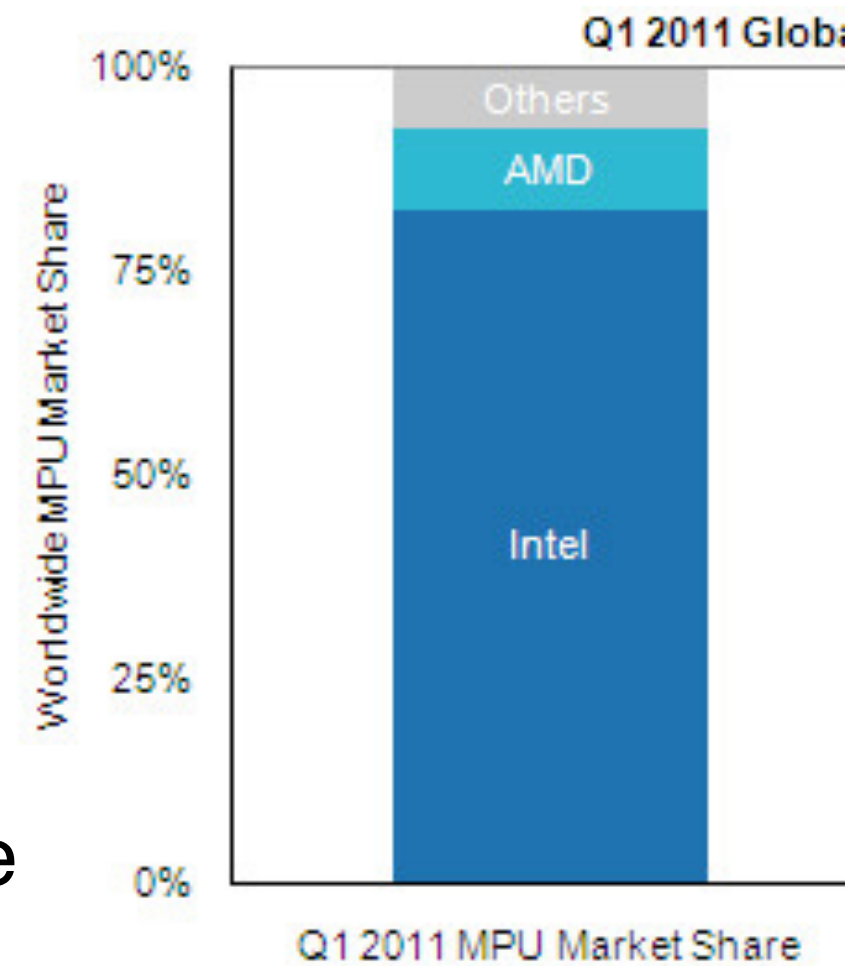
- Some basic concepts necessary to understand in more detailed manner current CPU architectures :
 - ✓ basic execution environment
 - ✓ few technology issues and their impact on the micro-processor architecture
 - ✓ need for parallelism :
 - ❖ for high-performance multi-media computing (SIMD),
 - ❖ but also more general parallel computing (thread-level parallelism)

Program for today

- Apply all this on the real example !
- What is better then **Intel CPUs** ?
 - ✓ Few infos on the company
 - ✓ Brief overview of the past and future Intel's micro-architectures
 - ✓ Few words on NetBurst micro-architecture (**past**) (Pentium4 micro-processors)
 - ✓ More detailed overview of Sandy/Ivy Bridge (**present**) micro-architectures (i3, i5 and i7 micro-processors)
 - ✓ Power and thermal management in recent micro-architectures

Intel Corporation

- Funded in 1968
- Largest and **highest valued semiconductor co** still today
- Mostly involved in CPUs, but they manufacture other chips too (motherboard chipsets, various controllers, memory, graphics, etc.)
- In 2012 (big company!):
 - ✓ **100k employees**
 - ✓ **55 billion \$ revenue**
 - ✓ **11 billion \$ net income**
- Major competitor → AMD
 - ✓ Intel : more then 80% market share



Intel's strategy from the start

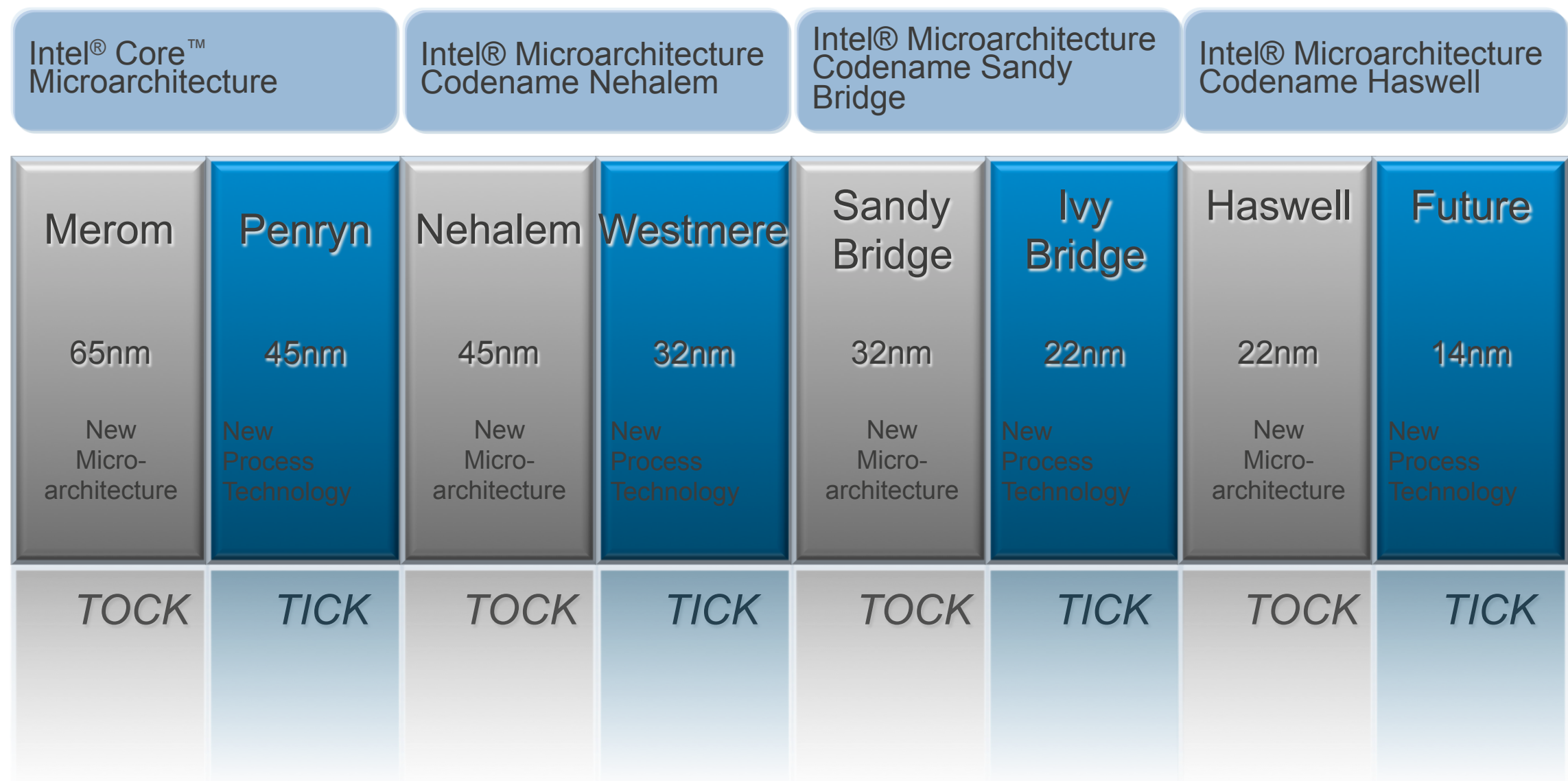
- Combine :
 - ✓ **Technology** development for (proprietary) IC manufacturing (AKA process technology)
 - ✓ **Circuit design** (i.e. IC architecture)
- Keep **both** aspects for themselves only !!!
 - This was true until now :
FPGAs vendors (Altera) might use their 14nm fab (**revolution!**)
- At some point in time Tech & Design were considered to be independent :
 - ✓ Split into : **Fab + Fabless companies**
 - Intel always ignored that with success
 - ✓ As we move along towards more and more advanced technologies, these two become very closely coupled
 - ✓ “Alone” approach not sustainable any more, hence the open-up

Intel's product line

- Focuses mostly on micro-processor market
- Targets mostly **high-performance** (and thus high-power) devices for laptops, desktops, servers, data-centers and super-computing
- In that field they have very little competition ...
- However one thing they missed is **low-power** applications !
- There was some trials to enter low-power market, but this ended with nothing great
 - ✓ ARM already created significant niche in this domain and established them as major player
 - ❖ Guess what is inside iPhone, iPad etc ...

Tick-Tock model: alternate process/design

- Combine Technology & Design in (extremely!) profitable way that alternate micro-architecture and technology development → maintains the product pipeline full all the time:)



Intel nomenclature disambiguation

- **Micro—architecture** : key differentiator, defines ISA, there are 2 major releases :
 - ✓ **x86** → laptop/desktop/server
 - depending on data/address path bit width
 - ❖ CISC
 - ❖ 32 bit and now 64 bit
 - ✓ **Itanium** → servers/high-performance computing
 - ❖ VLIW based architecture
 - ❖ remained low-volume
 - ❖ still in the roadmap
- **Micro-architecture code name**
 - ✓ Each of the above will have a specific code name (e.g. **NetBurst** or **Nehelem**)
- **Brand name** — this what you buy in the end of the day
 - ✓ **Pentium 4** (NetBurst) or **i7** (Nehelem)

**From 8086
to planned future generations**

- **8086** — First x86 processor
- **186** — Included a DMA, interrupt controller, timers, and chip select logic
- **286** — First x86 processor with protected mode
- **i386** — **First 32-bit** x86 processor
- **i486** — Intel's second-generation of 32-bit x86 processors, included **built in floating point unit and pipelining**
- **P5** — Original Pentium microprocessors
- **P6** — Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, and Pentium III Xeon
 - ✓ Pentium M — Updated version of Pentium III's P6 micro-architecture designed from the ground up for **mobile computing**
 - ✓ Enhanced Pentium M — Updated, dual core version of the Pentium M micro-architecture used in Core microprocessors
- **NetBurst (2001)**
 - ✓ Used in Pentium 4, Pentium D, and some Xeon microprocessors
 - ✓ Commonly referred to as P7 although its internal name was P68 (P7 → Itanium)
 - ✓ Later revisions were the first to feature Intel's **x86-64 architecture**

Single-core

Multi-core

Planar

- **Core (2006)** — Re-architected P6-based micro-architecture used in Core2 and Xeon microprocessors, built in 65nm process
 - ✓ **Penryn**
 - ❖ 45nm shrink of the Core micro-architecture with larger cache, faster FSB and clock speeds, and SSE4.1 instructions.
- **Nehalem (2008)**
 - ✓ 45nm process → Core i7, Core i5, Core i3 microprocessors
 - ✓ Incorporates the off-chip memory controller into the CPU die.
 - ❖ **Westmere** 32nm shrink of the Nehalem micro-architecture with several new features
- **Sandy Bridge (2011)** — 32 nm process
 - Core i7, Core i5, Core i3 2nd generation CPUs
 - ✓ **Ivy Bridge (2012)**
 - ❖ 22 nm shrink of the Sandy Bridge micro-architecture

Multi-core

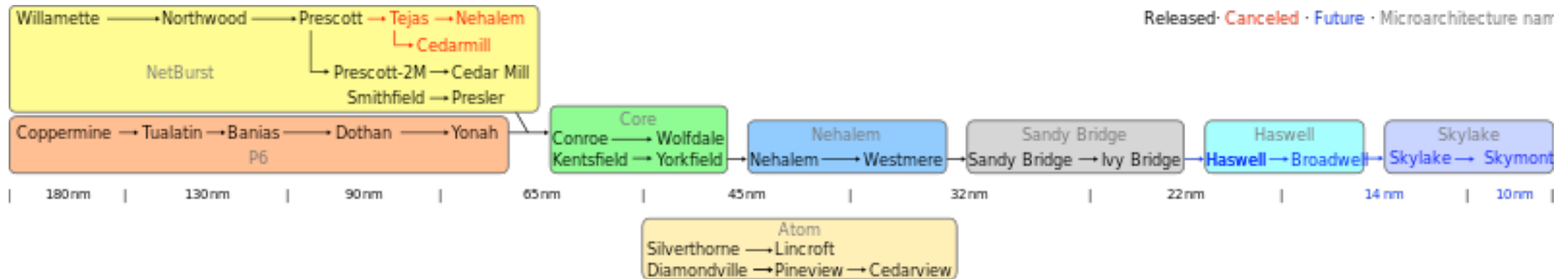
Multi-Gate

- **Haswell** — Future Intel micro-architecture, expected 2013, 22nm process
 - ✓ **Broadwell** — 14nm shrink of the Haswell micro-architecture
 - ❖ expected around 2014 (formerly called Rockwell)
- **Skylake** — Future Intel micro-architecture, based on a 14nm process
 - ✓ **Skymont** — 10 nm shrink of the Skylake micro-architecture
- **Larrabee** — Multi-core in-order x86-64 updated version of P5 micro-architecture
 - ✓ wide SIMD vector units and texture sampling hardware for use in graphics
 - ✓ cores derived from this micro-architecture are called MIC (Many Integrated Core)
- **Bonnell** — Low-power, in-order micro-architecture for use in Atom processors
 - ✓ **Saltwell** — 32 nm shrink of the Bonnell micro-architecture.

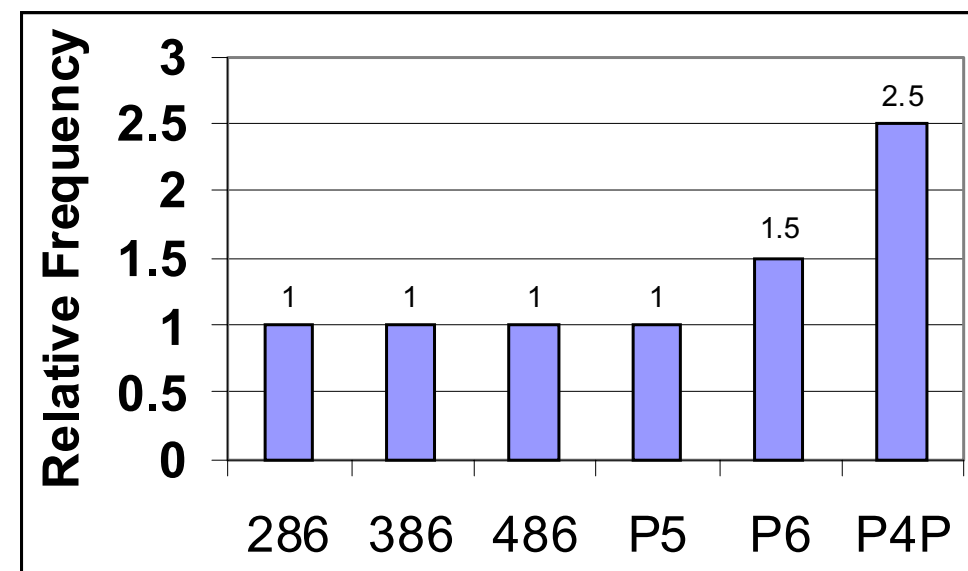
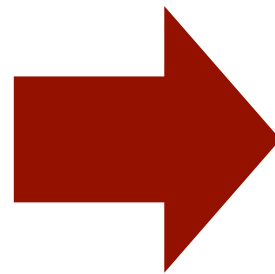
Delayed!!!

Future : advanced
tech nodes

Cores roadmap, micro-architectures



Microarchitecture	Pipeline stages
P5 (Pentium)	5
P6 (Pentium Pro)	14
P6 (Pentium 3)	10
NetBurst (Willamette)	20
NetBurst (Northwood)	20
NetBurst (Prescott)	31
NetBurst (Cedar Mill)	31
Core	14
Bonnell	16

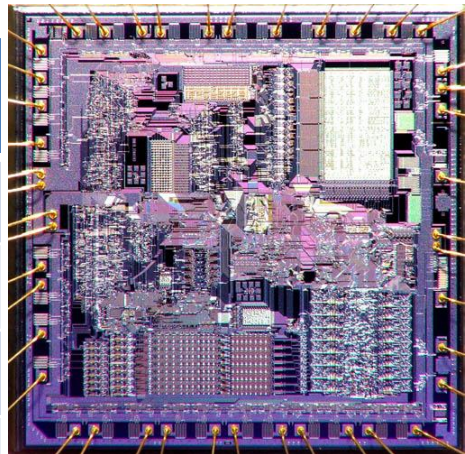


You now know why !

Intel IA32

8086

Proc	Intro	MHz(init)	Trans/Die	Registers	Ext Bus	Max Addr	Cache
8086	1978	8	29K	16GP	16	1MB	None
286	1982	12.5	134K	16GP	16	16MB	None
386	1985	20	275K	32GP	32	4GB	8KB on chip
486	1989	25	1.2M	32GP/80FPU	64	4GB	L1:8KB
Pentium	1993	60	3.1M	32GP/80FPU	64	4GB	L1:16KB
Pent Pro	1995	200	5.5M	32GP/80FPU	64	64GB	L1:16KB L2:256KB or 512KB
Pent II	1997	266	7M	32GP/80FPU 64MMX	64	64GB	L1:16KB L2:256KB or 512KB
Pent III	1999	500	8.2M	32GP/80FPU 64MMX/128XMM	64	64GB	L1: 32KB L2: 512KB
Pent IV	2000	1500	42M	32GP/80FPU 64MMX/128XMM	64	64GB	L1:12Kuops(trace) 8KB data L2:256KB



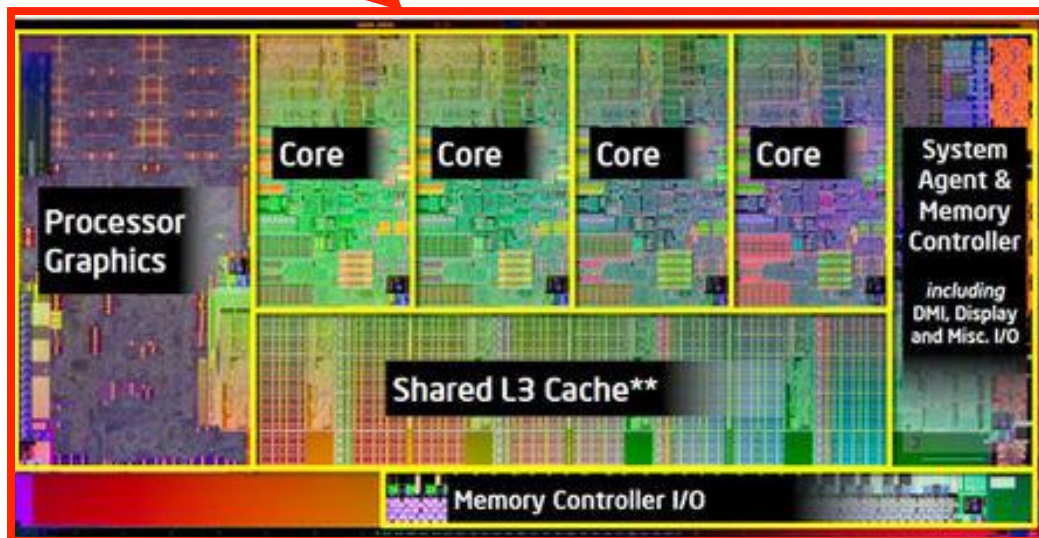
Pentium4



Operand size & register file evolution

IA32 and Intel64

Proc	Intro	MHz(init)	Trans/Die	Registers	Ext Bus	Max Addr	Cache
Pent M	2003	900	140M	32GP/80FPU 64MMX/128XMM	64	64GB	L1: 32KB L2: 1M
Core (Duo)	2006	1500 1600	228M	32GP/80FPU 64MMX/128XMM	64	64GB	L1: 32KB L2: 2M
Core 2 (Duo/Quad)	2006	1600 2000	410M (2) 820M (4)	32GP/80FPU 64MMX/128XMM	2x64	1TB	L1: 32KB L2: 2M/4M or 6/12MB
Nehalem (2-8 cores)	2008	2660	731M	32GP/80FPU 64MMX/128XMM	3x64	256TB	L1:32KB L2:256KB L3: 4MB-12MB
Sandy Bridge (2-8 cores)	2011	2800	1.16G (4) 2.27G (8)	32GP/80FPU 64MMX/128XMM 256YMM	4x64	256TB	L1:32KB L2:256KB L3: 8MB-20MB



As much L2 as total
addressable memory of 8086

**Note the bit-width evolution for
the off-chip DRAM**

Intel's future ?

- Not very clear ...
- Desktop will not be there for a very long time
- Market for high-performance micro-processors will most likely dramatically shrink in the next few years :
 - ❖ It has been already some time that mobile market (smart-phones + tablets) surpassed PC market
- **Will data-center/server/high-performance computing be enough to feed the monster ?**
 - ✓ Very difficult to say ... but we can have reasonable doubts
 - ✓ Sign for the winds of change is the fab opening
 - ✓ After all they did cut a deal with Apple ...
- Anyhow for 14, 10 and maybe 7nm they are going to be there for sure ...

NetBurst micro-architecture (Pentium4)

Evolution

- Micro-architecture that lasted from 2000 to 2006 ! (long ...)
- Tick-tock principal applied but more then this ...
- Important micro-architectural changes introduced throughout different revisions

Revision (process)	Brand	pipe depth
Willamette (180nm)	Celeron, Pentium 4	20
Northwood (130nm)	Celeron, Pentium 4, Pentium 4 HT	20
Gallatin (130nm)	Pentium 4 HT Extreme Edition, Xeon	20
Prescott (90nm)	Celeron D, Pentium 4, 4 HT, Extreme	31
Cedar Mill (65nm)	Celeron D, Pentium 4	31
Smithfield (90nm)	Pentium D	31
Presler (65nm)	Pentium D	31
42 million transistors, six aluminum metal layers, die=217 mm ² , 55W@1.5GHz, 3.2 GB/second system bus		

Global overview

- NetBurst micro-architecture → significant upgrade from P6
 - ✓ **Single core**, 20 stage pipeline (31 for Prescott) — **hyper-pipeline**
 - ❖ enables much higher clock rates
 - ➔ early beginnings of **thermal issues** — abandon of certain high-speed revisions (see road-map → beginning of multi-core era)
 - ❖ two cycles to drive result across the chip!
 - ✓ Seven integer execution units against 5 for P6
 - ❖ **additional integer ALU plus additional address unit**
 - ✓ Further extensions of SIMD (MMX + SSE→SSE2)
 - ✓ **SMT (HyperThreading) from Prescott revision**
- Uses register renaming
- Different parts of the Pentium4 run at different clock speeds
- An aggressive integer ALU operates at twice the clock rate

Architecture overview

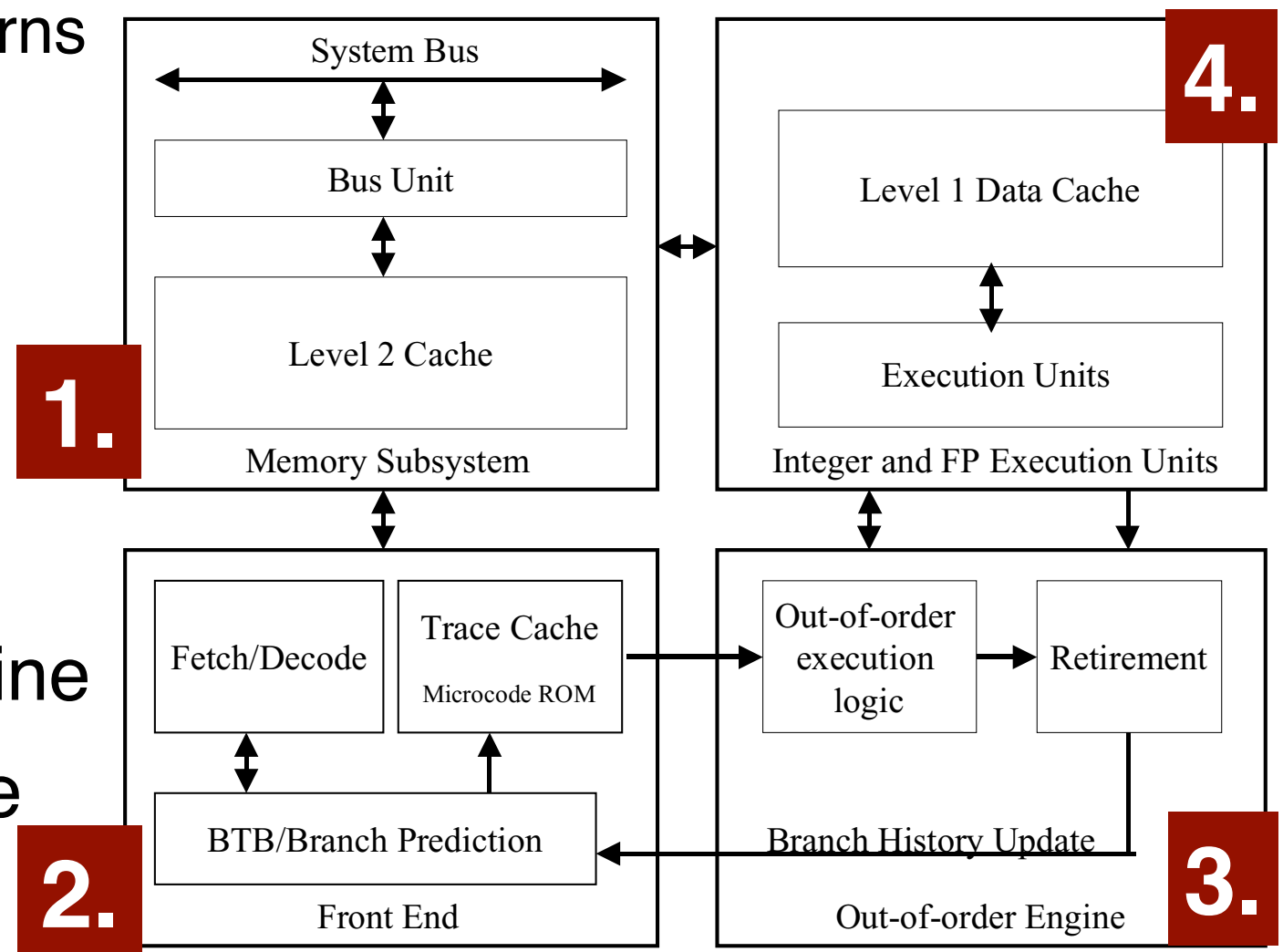
1. Bus unit and unified (data & instructions) L2 cache memory

- ✓ 256kB 8-way set-associative cache with 128B per cache line
- ✓ write-back strategy
- ✓ associated with the L2 cache is a hardware prefetcher
 - ❖ monitors data access patterns and prefetches data automatically into the L2
 - ❖ attempts to stay 256 bytes ahead of the current data access locations

2. In-order front-end

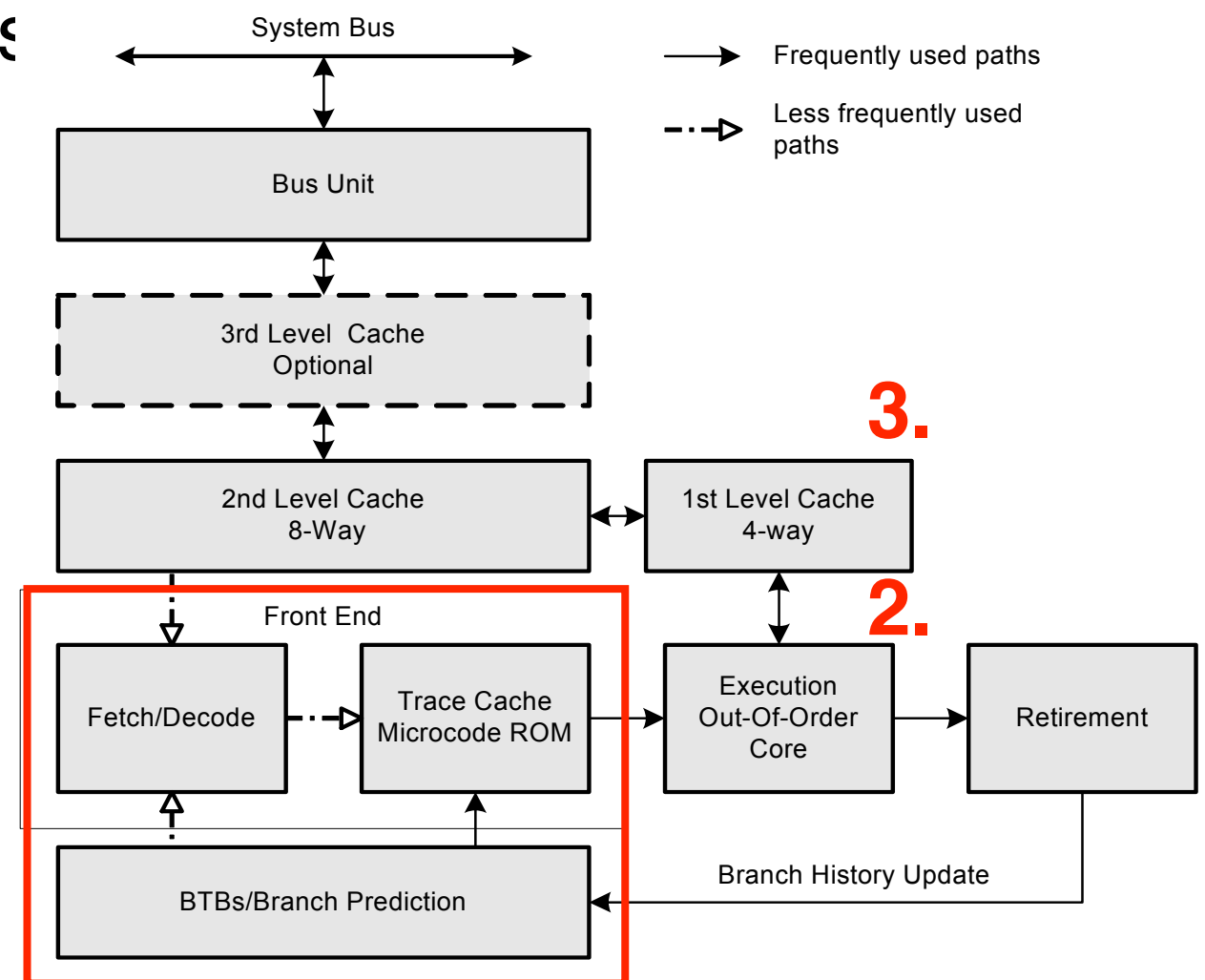
3. Out-of-order execution engine

4. Integer & FP execution core with multiple ALUs



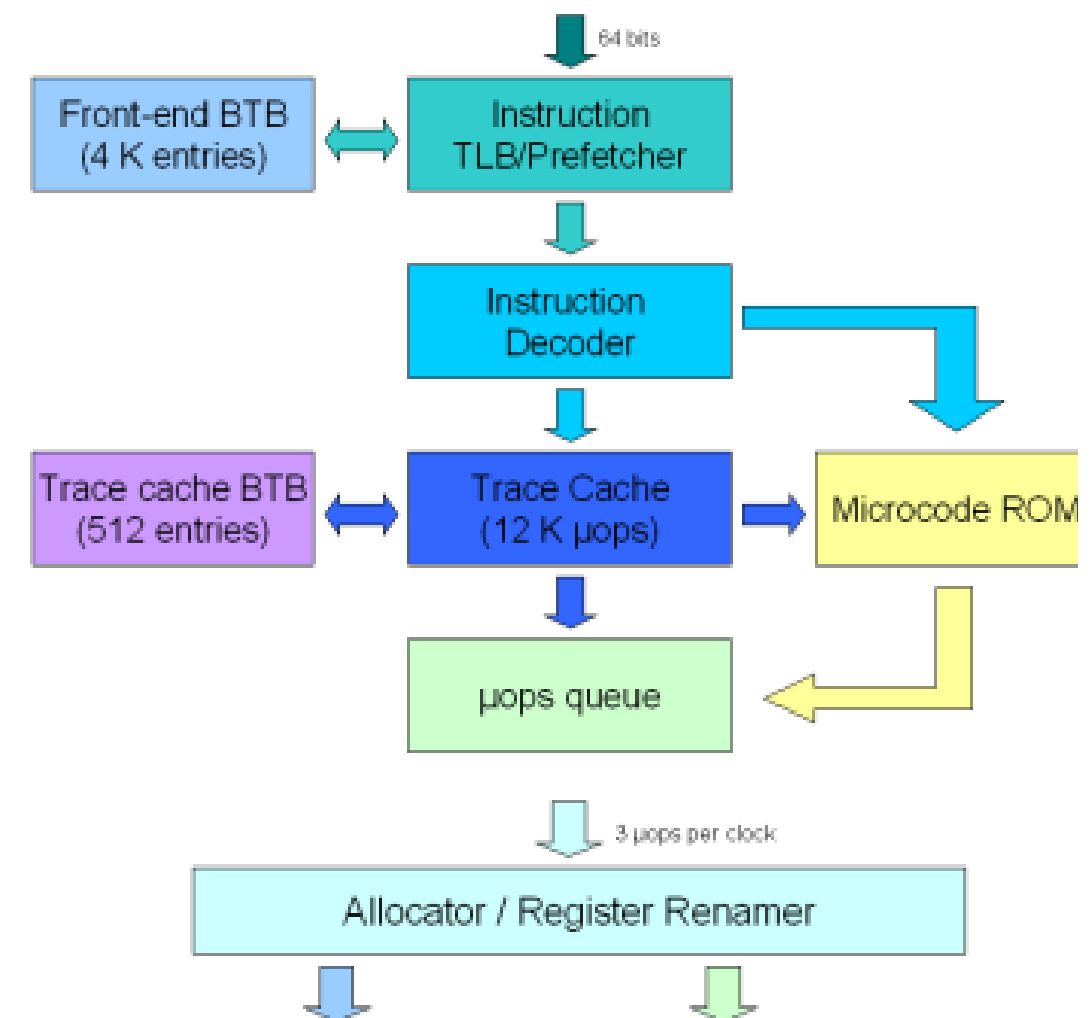
Front-end

- In-order front-end (**1.**) feeds the Out-of-Order (OO) Execution Core (**2.**) connected to L1 data cache (**3.**)
- Prefetches instructions that are likely to be executed
- Fetches instructions that have not already been prefetched
- Decodes instructions into u-ops
- Generates microcode for complex instructions and special-purpose code (CISC)
- Delivers decoded instructions from the execution trace cache (Instruction L1)
- Predicts branches using advanced prediction algorithm



Front-end operation

- Instructions are most of the time loaded from **Trace Cache** (instruction L1) : instructions are already decoded here !
- If we have cache miss, the instruction will be loaded from L2 and decoded into micro-ops
- Instruction **Translation Lookaside Buffer** (ITLB) translates the virtual memory pointer into physical addresses needed to access the L2 cache
- Branch prediction using Branch Target Buffer (BTB)
- If the instruction is complex, the correct micro-ops sequence might come from Microcode ROM
- micro-ops are queued for out-of-order execution

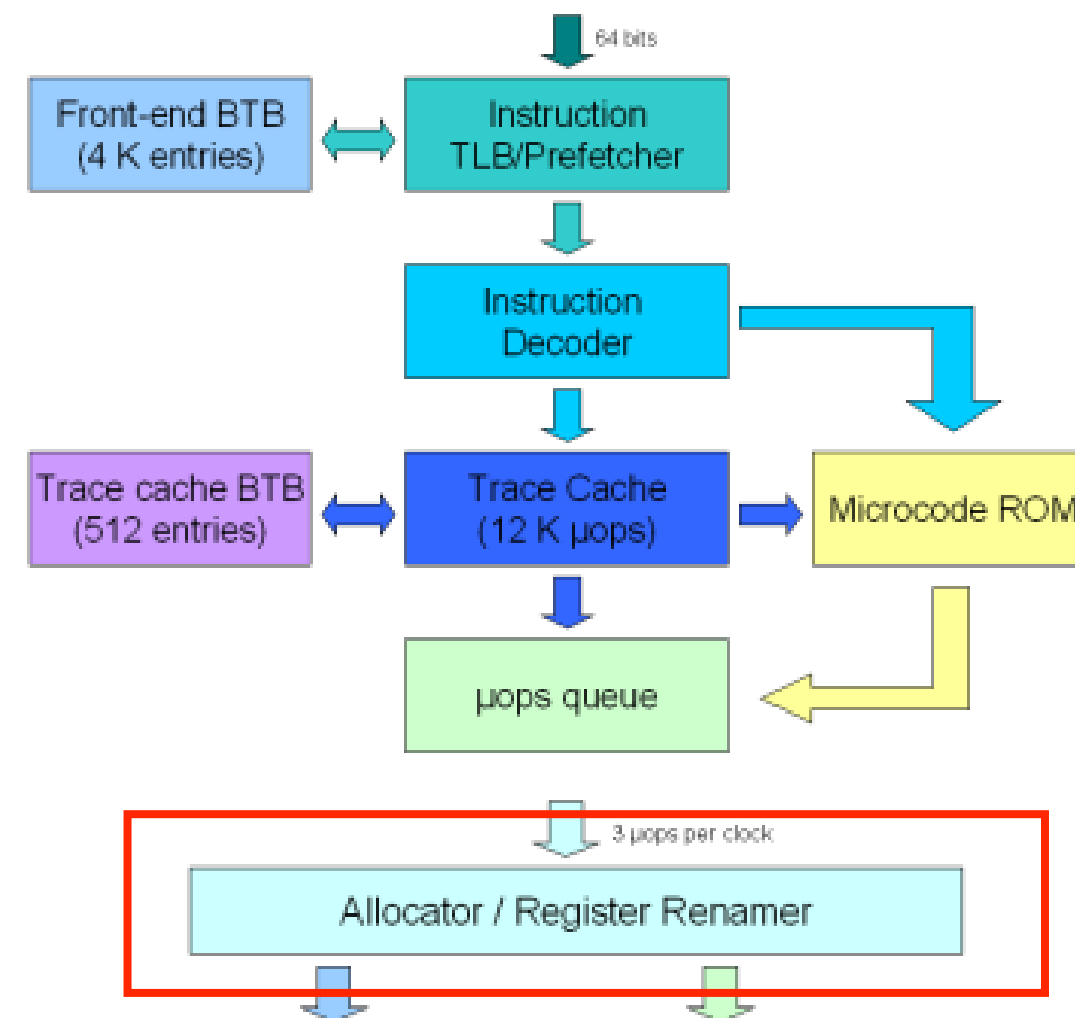


Branch prediction

- **Hardware instruction prefetching** logic associated with the front-end BTB fetches IA-32 instruction bytes from the L2 cache that are predicted to be executed next
- The fetch logic attempts to keep the instruction decoder fed with the next IA-32 instructions the program needs to execute
- This instruction prefetcher is guided by the branch prediction logic (branch history table and branch target buffer listed here as the front-end BTB) to know what to fetch next
- Branch prediction allows the processor to begin fetching and executing instructions long before the previous branch outcomes are certain
- The front-end branch predictor is large to capture most of the branch history information for the program
- If a branch is not found in the BTB, the branch prediction hardware **statically predicts** the outcome of the branch based on the direction of the branch displacement

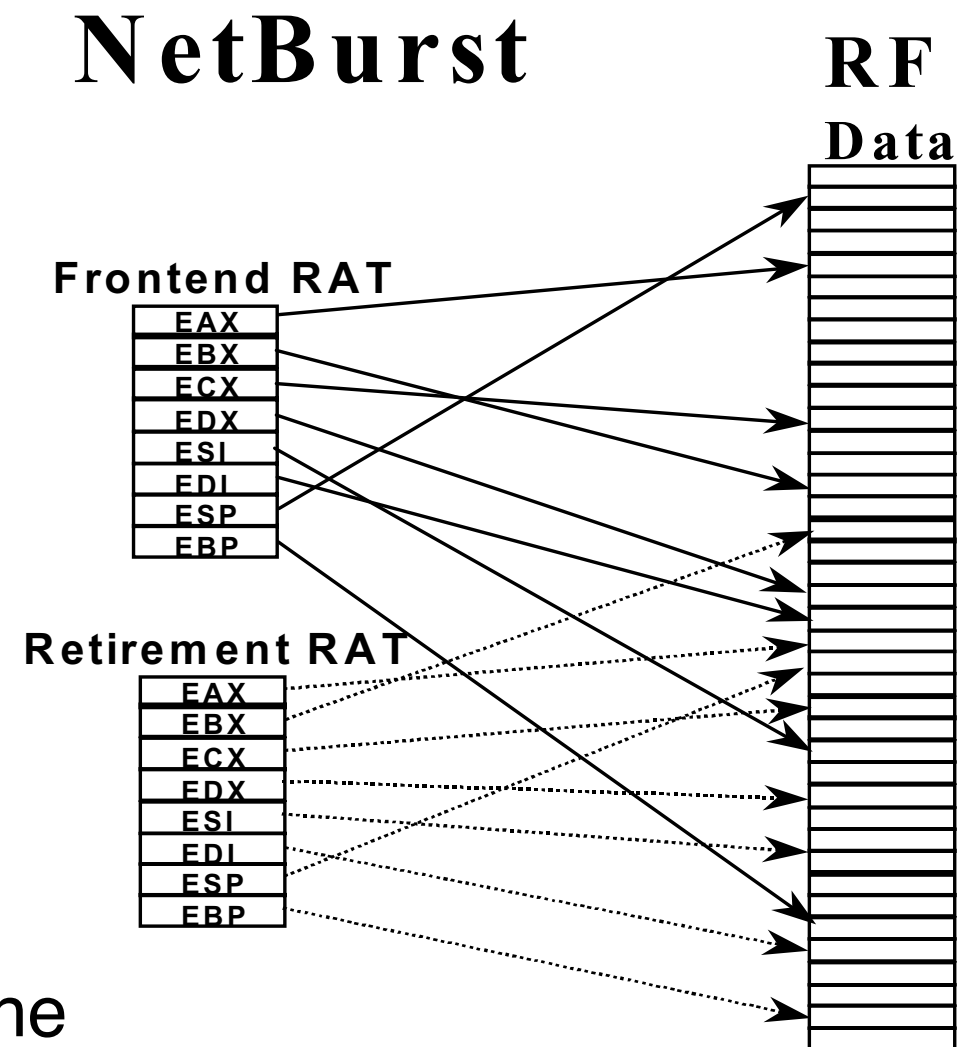
Allocator

- Out-of-order execution engine has buffers to perform its re-ordering, tracking, and sequencing operations
- **Allocator logic allocates buffers** needed by micro-op to execute
- If a needed resource, such as a register file entry, is unavailable for one of the three uops coming to the Allocator this clock cycle, the Allocator will stall this part of the machine
- When the resources become available the Allocator assigns them to the requesting uops and allows these satisfied uops to flow down the pipeline to be executed
- The Allocator also allocates one of the 128 integer or floating-point register entries for the result data value of the uop, and possibly a load or store buffer used to track one of machine pipeline



Register renaming

- The register renaming logic renames the logical IA-32 registers (e.g. EAX) on a **128-entry physical register file**
- This allows the small, 8-entry, architecturally defined IA-32 register file to be dynamically expanded to use the 128 physical registers
- Renaming process removes false conflicts caused by multiple instructions creating their simultaneous but unique versions of a register such as EAX
- There could be dozens of unique instances of EAX in the machine pipeline at one time!
- The renaming logic remembers the most current version of each register, such as EAX, in the Register Alias Table (RAT) so that a new instruction coming down the pipeline can know where to get the correct current instance of each of its input operand registers

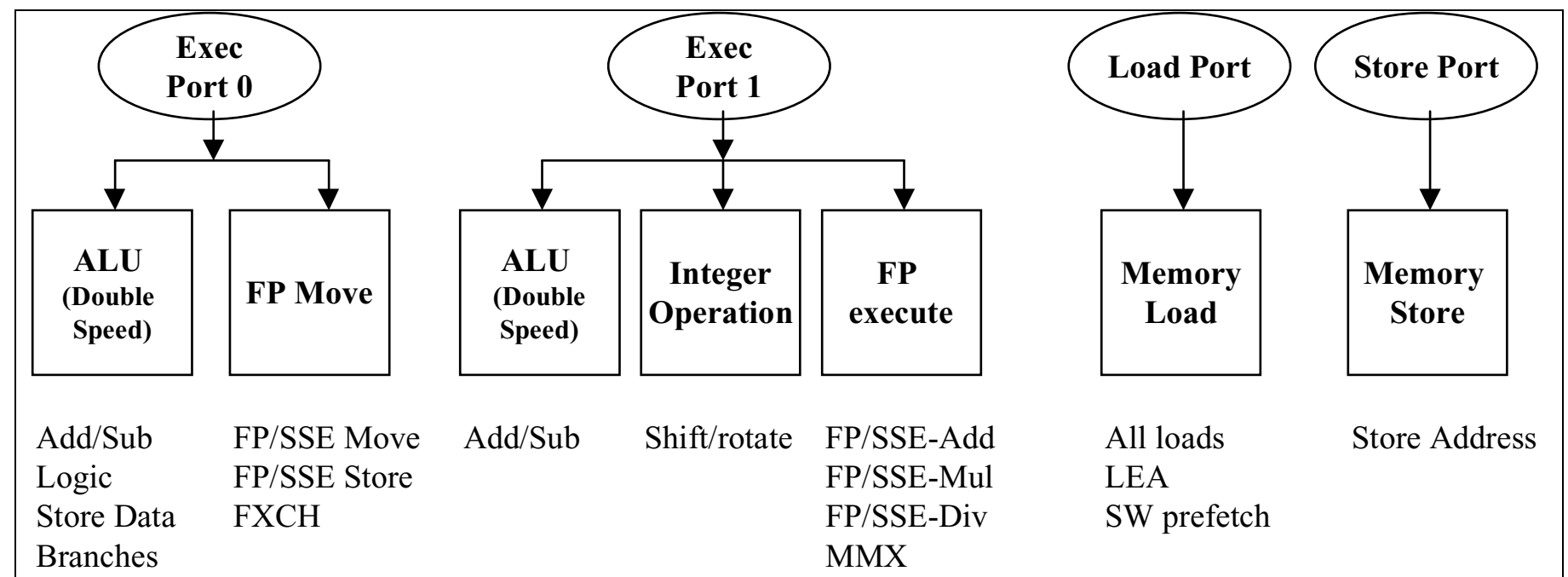


Execution core

- Specialized 4 port dispatch unit with 7 ALUs → **super-scalar**
- Floating-Point (x87), MMX, SSE (Streaming SIMD Extension), and SSE2 (Streaming SIMD Extension 2) operations are executed by the two floating-point execution blocks
- MMX instructions are 64-bit packed integer SIMD operations that operate on 8, 16, or 32-bit operands
- SSE instructions are 128-bit packed IEEE single-precision FP floating-point operations

- **SSE2**

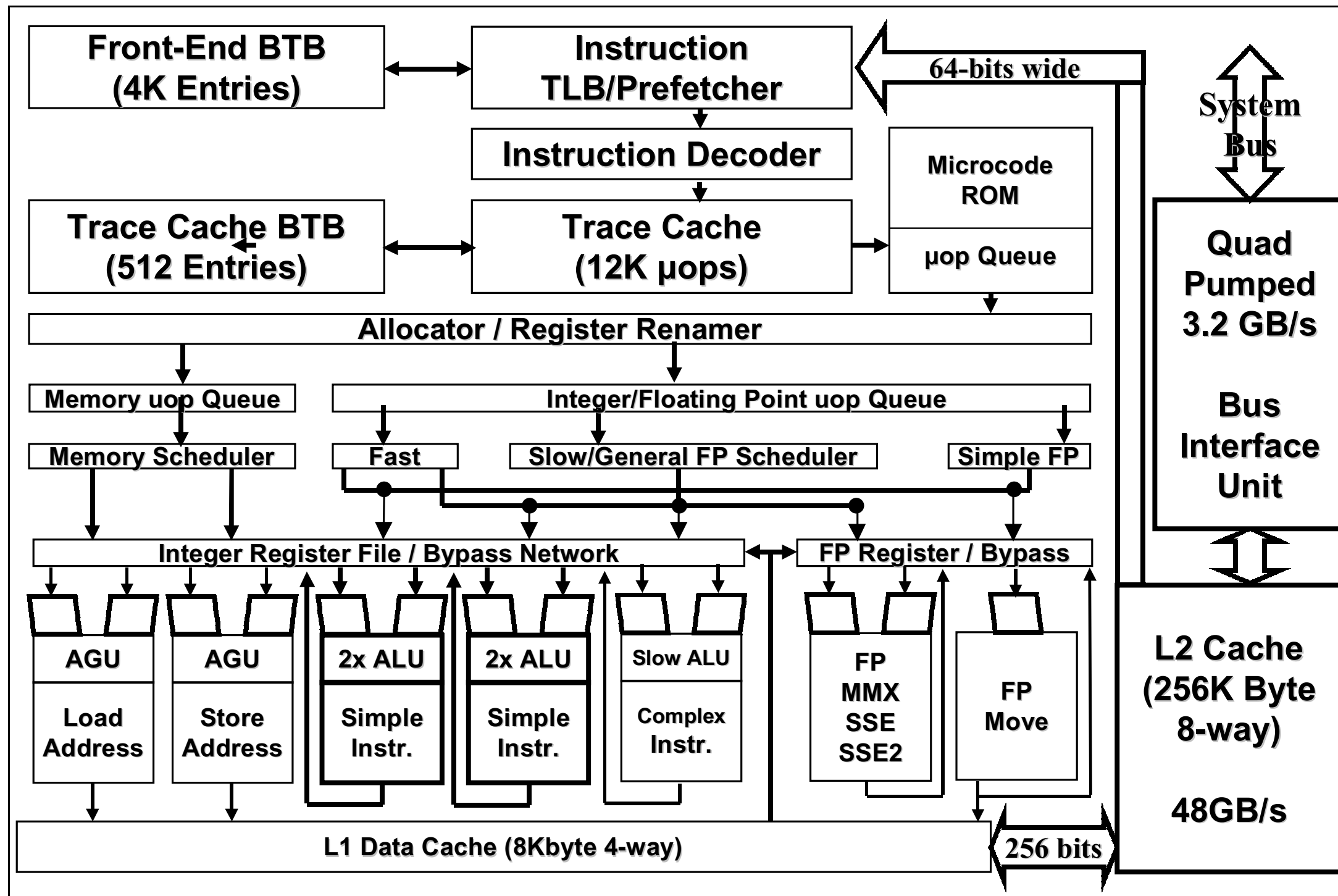
✓ 128-bit single and double precision FP



micro-op scheduling

- The micro-op schedulers determine **when** a micro-op is ready to execute by tracking its input register operands
- **They reorder instructions** to execute as soon as they are ready:
 - ✓ uop queues : store in FIFO fashion but read in out-of-order
 - ❖ For memory operations
 - ❖ For non-memory operations (computation)
- Schedulers are tied to **four different dispatch ports**
- There are two fast exec units dispatch ports: Port0 & Port1
 - ✓ **fast**: they can dispatch up to two operations each main processor clock cycle
- Multiple schedulers share each of these two dispatch ports
- Other schedulers can only schedule once per main processor clock cycle
- There is also a load and a store schedule

Complete CPU



ELEC-H-473
Microprocessor architecture
Intel x86 micro-architecture
Ivy/Sandy Bridge

Lecture 10,11

Dragomir MILOJEVIC
dragomir.milojevic@ulb.ac.be

Sandy & Ivy Bridge Micro-architectures

Overview

- **Sandy Bridge (2011)**

- ✓ Used in Core i7, Core i5, Core i3 second generation microprocessors (succeeding Core architecture)
- ✓ With 32nm process
 - **Tick : new micro-architecture**

- **Ivy Bridge (2012)**

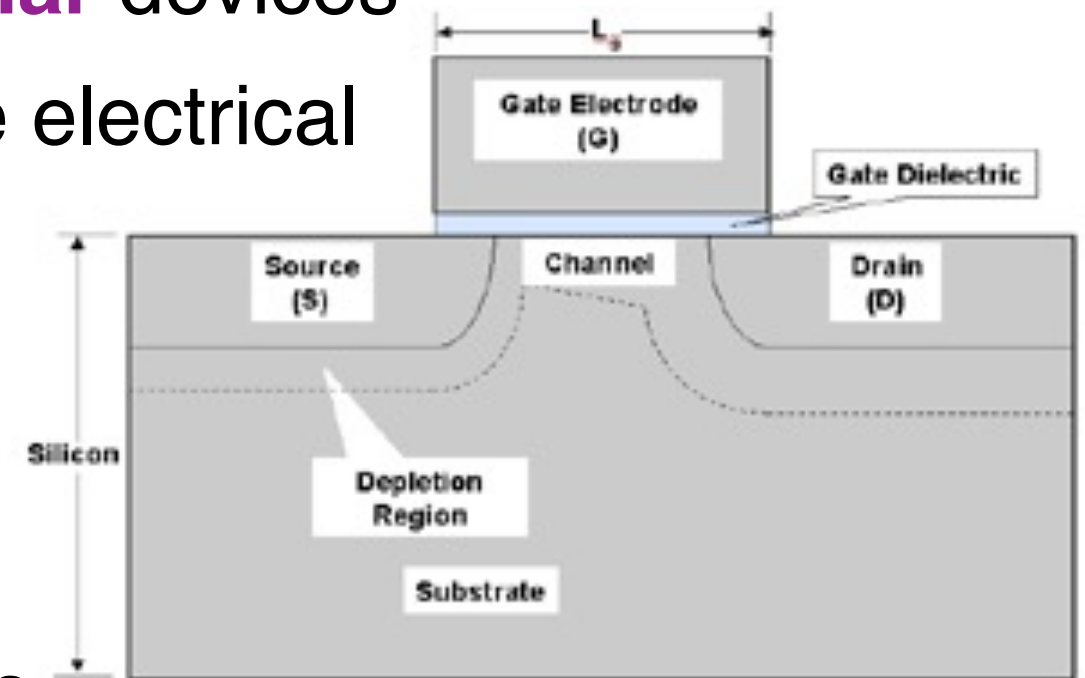
- ✓ 22 nm shrink of the Sandy Bridge micro-architecture
 - **Tock : new technology**

- This 22nm technology node brought new technology feature :

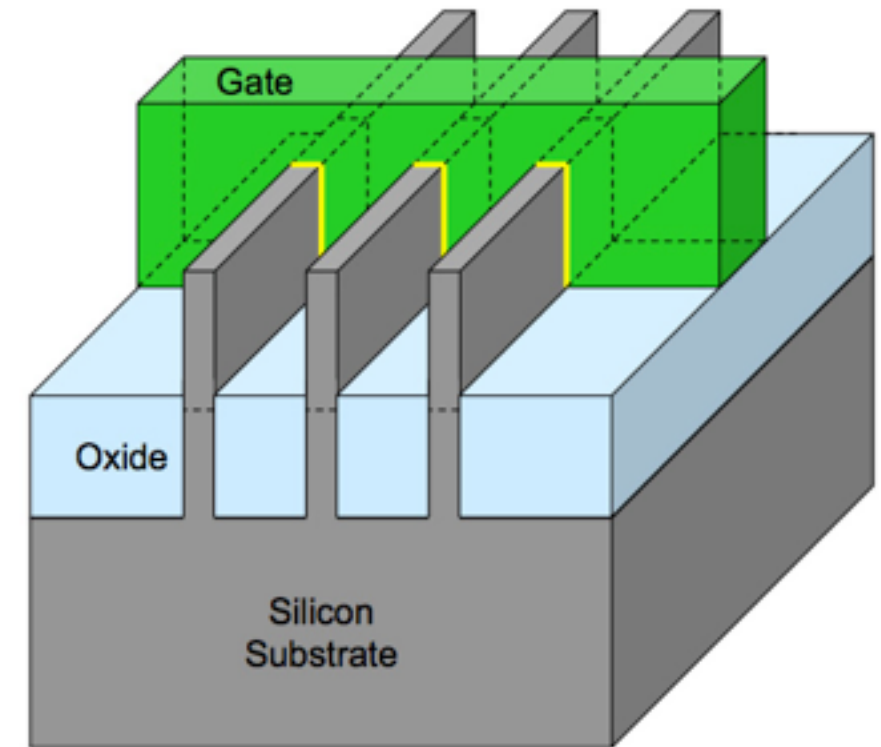
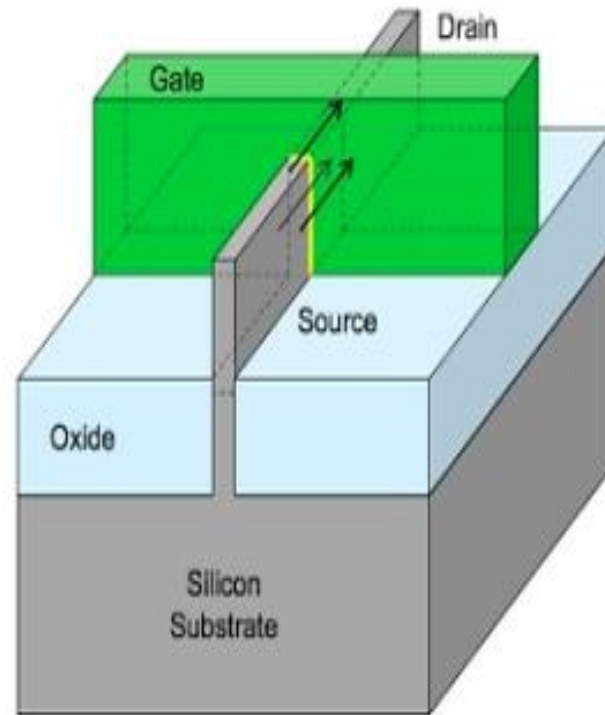
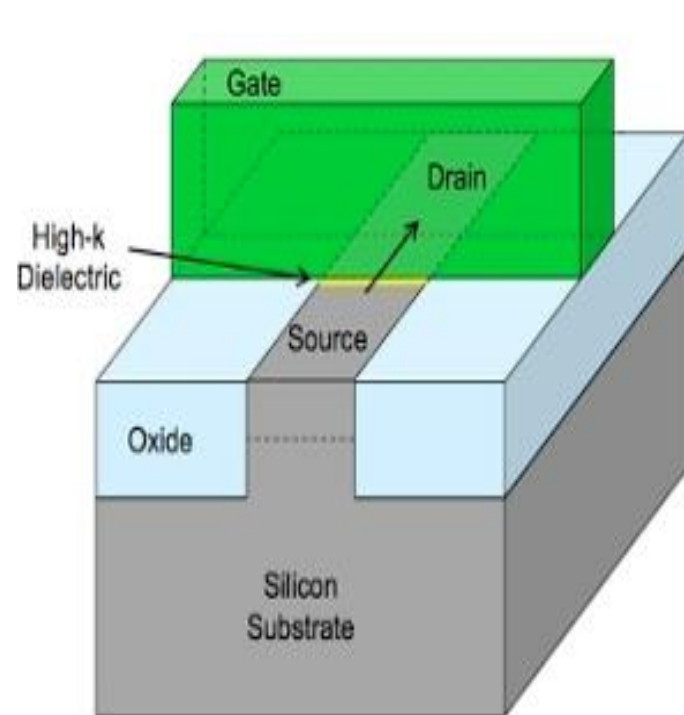
- ✓ **Multi-gate transistors**
- ✓ Main gain : performance per watt
 - ❖ 0.5X power for the same performance

Planar transistor vs. 3D transistor

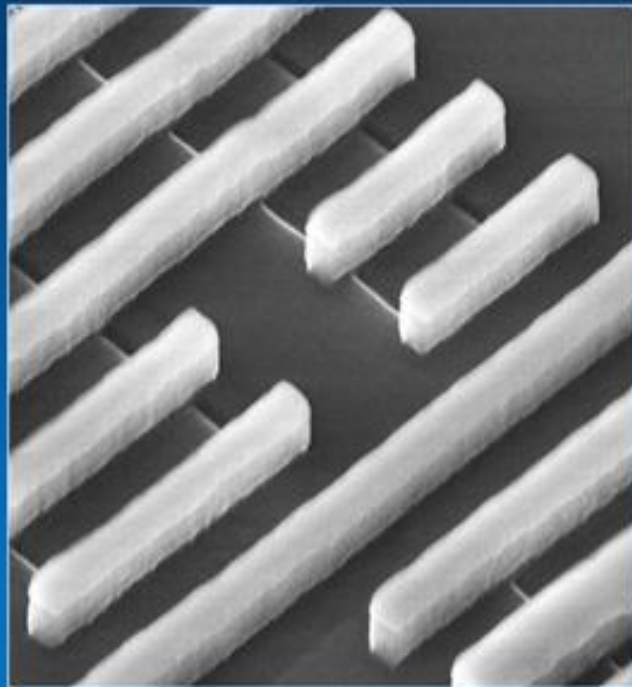
- Standard CMOS transistors are **planar** devices
- Shrinking starts to be a trouble since electrical properties are coupled to geometry
- At one point in time it is complicated to reduce all **areas** !
- 3D transistor concept in 1989
- First 3-gate transistor in 2002 & 2006
- **3-D transistors (multi-gate, or FinFETs)**, a block material across the top of the channel
- A tri-gate transistor has a channel with three dimensions
- The flow of electricity is on all three sides
- It reduces transistor size on silicon die to the width of the fin while still being a long enough gate for a good signal



Planar vs. multi-gate



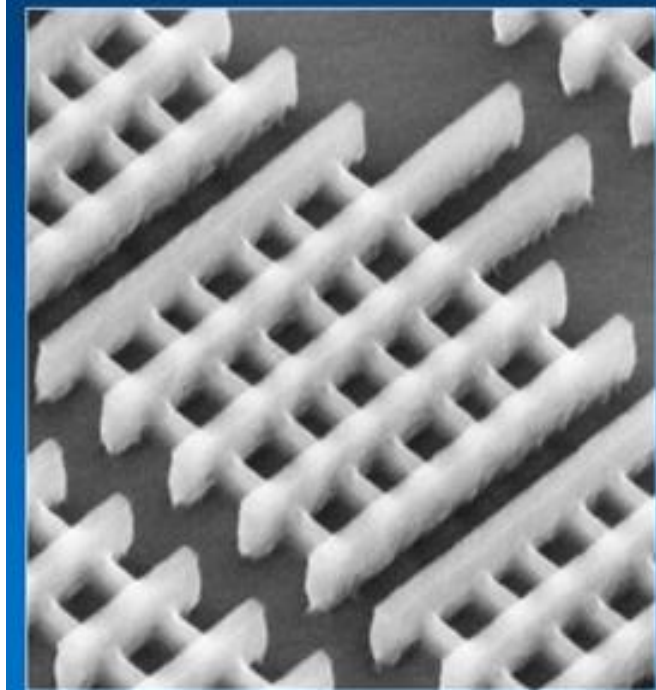
32 nm Planar Transistors



**Channel
grows into
vertical direction**

[Link](#)

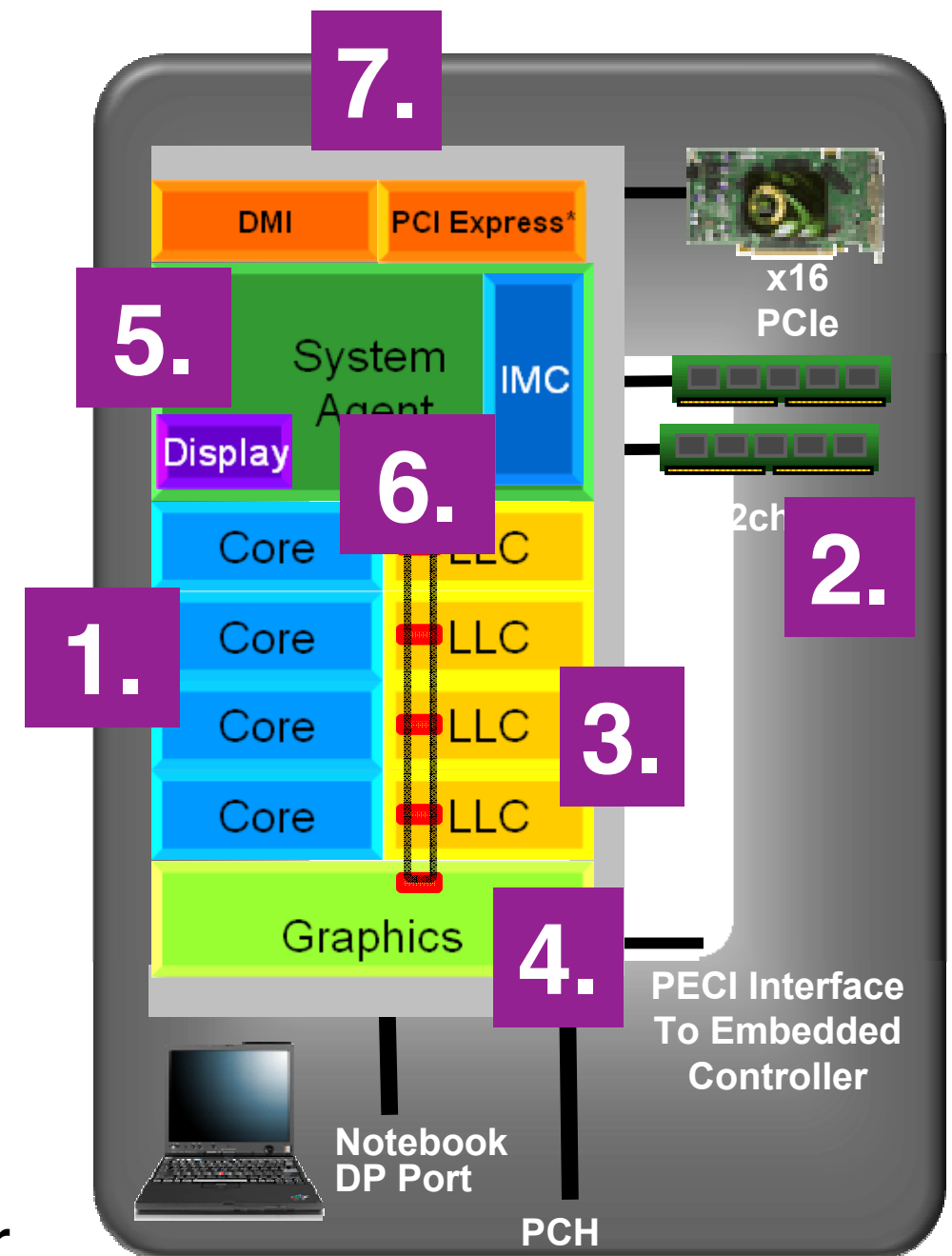
22 nm Tri-Gate Transistors



SandyBridge : micro-architecture overview

CPU → SoC : global overview

1. 2-8 physical **cores** implementing Intel's SMT → **HyperThreading**
 - each physical core correspond to 2 logical cores (2 threads per core)
 - **TurboBoost Technology**
2. Integrated memory controller
2 channel DDR3
(up to 4 channels of DDR3@1600MHz)
3. High bandwidth Least Level Cache (LLC) → L3
4. **SoC** interconnect
5. Embedded high-performance graphics processor (GPU)
6. System Agent — Overall SoC controller
7. SoC IO



Hyper-threading in action

Logical cores

Physical cores

The screenshot shows the macOS Hardware Overview window. The menu bar at the top displays a '2' in the system status area, which is circled in red and labeled 'Logical cores'. The Hardware Overview window lists the following information:

Hardware Overview:	
Model Name:	MacBook Pro
Model Identifier:	MacBookPro6,2
Processor Name:	Intel Core i7
Processor Speed:	2.66 GHz
Number Of Processors:	1
Total Number Of Cores:	2
L2 Cache (per core):	256 KB
L3 Cache:	4 MB
Memory:	8 GB
Processor Interconnect Speed:	4.8 GT/s
Boot ROM Version:	MBP61.0057.B0C
SMC Version (system):	1.58f16
Serial Number (system):	W801929EAGZ
Hardware UUID:	006E2966-4CE3-5F85-B38B-9E73CD1E1B54
Sudden Motion Sensor:	State: Enabled

The 'Total Number Of Cores' value of 2 is circled in red and labeled 'Physical cores'.

How to assign a given thread to a core?

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
HANDLE *m_threads = NULL;
DWORD_PTR WINAPI threadMain(void* p);

DWORD_PTR GetNumCPUs() {                                // get n° of logical cores
    SYSTEM_INFO m_si = {0, };
    GetSystemInfo(&m_si);
    return (DWORD_PTR)m_si.dwNumberOfProcessors;
}
int wmain(int argc, wchar_t **args) {
    DWORD_PTR c = GetNumCPUs();
    m_threads = new HANDLE[c];
    for(DWORD_PTR i = 0; i < c; i++) {
        DWORD_PTR m_id = 0;
        DWORD_PTR m_mask = 1 << i;
        m_threads[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)threadMain, (LPVOID)i,
NULL, &m_id);                                           // create threads
        SetThreadAffinityMask(m_threads[i], m_mask);    // set thread affinity
        wprintf(L"Creating Thread %d (0x%08x) Assigning to CPU 0x%08x\r\n", i,
(LONG_PTR)m_threads[i], m_mask);
    }
    return 0;
}
DWORD_PTR WINAPI threadMain(void* p) {return 0;}
```

Turbo Boost

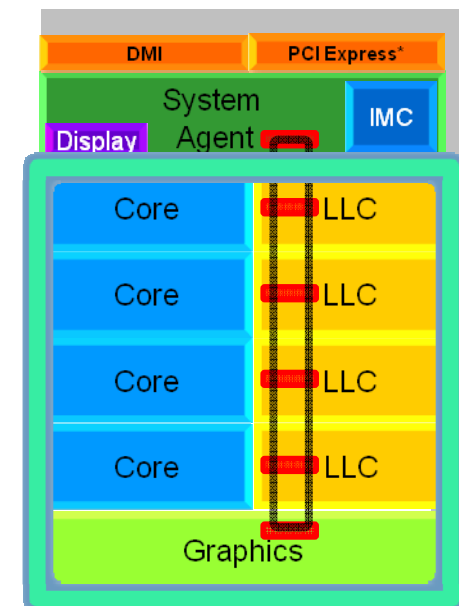
- Le CPU has few operating points in terms of Vdd, F
 - ✓ Depending on the Vdd we can chose F
 - ❖ Higher Vdd → higher operating F
 - ✓ But both have impact on power since :
 - ❖ $P = C \times Vdd^2 \times F$
- **TurboBoost : allows cores to run much faster then nominal operating F for a very short amount of time**
- Idea behind : uses thermal inertia of the IC + package !
 - ✓ Much slower then processing bursts
 - ✓ Uses whole IC as kind of a heat spreader/sink

DDR IO — solving memory bottleneck

- Embedded DDR-DRAM Controller
- **2 (up to 4) independent channels for DDR3 & DDR3L support**
 - ✓ Low voltage DDR3 (DDR3L) support for mobile
- DDR Over-clocking
 - ✓ support for up to 2800 MT/s (up from 2133)
 - ✓ Finer grain steps in adjusting frequency
 - ✓ Added 200 MHz
- DDR I/O embedded **power gating**
 - ✓ Power off DDR I/O when idle

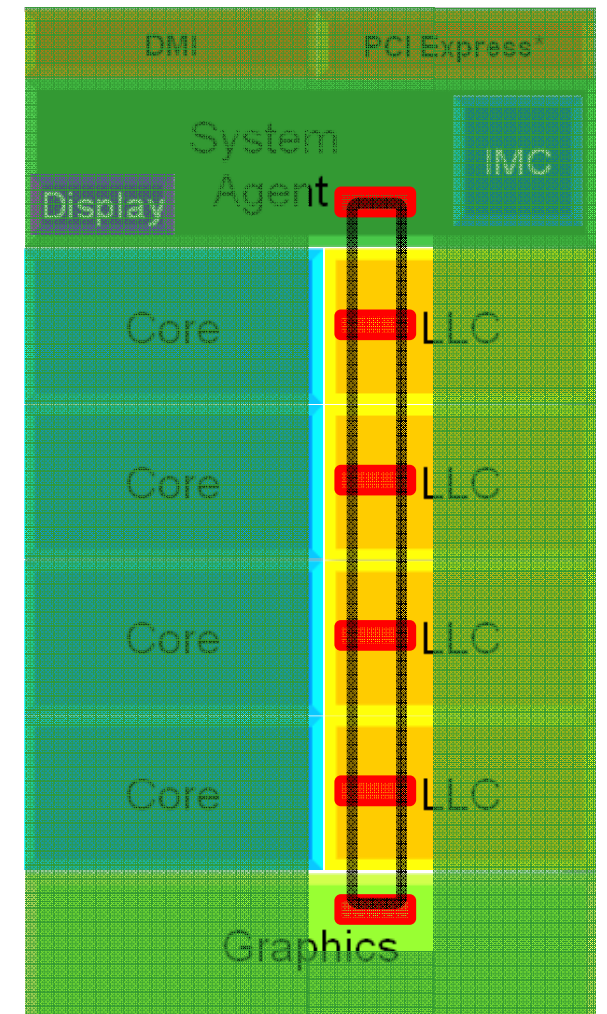
LLC

- High graphics performance, DRAM power savings, more DRAM BW available for cores
- LLC is shared among all cores, graphics and media
 - ✓ Graphics driver controls which streams are cached/coherent
- Any agent can access all data in the LLC, independent of who allocated the line, after memory range checks
- Controlled LLC way allocation mechanism to prevent thrashing between Core/graphics
- Multiple coherency domains
 - ✓ IADomain - cores (Fully coherent via cross-snoops)
 - ✓ Graphic domain (Graphics virtual caches, flushed to IA domain by graphics engine)
 - ✓ Non-Coherent domain (Display data, flushed to memory by graphics engine)



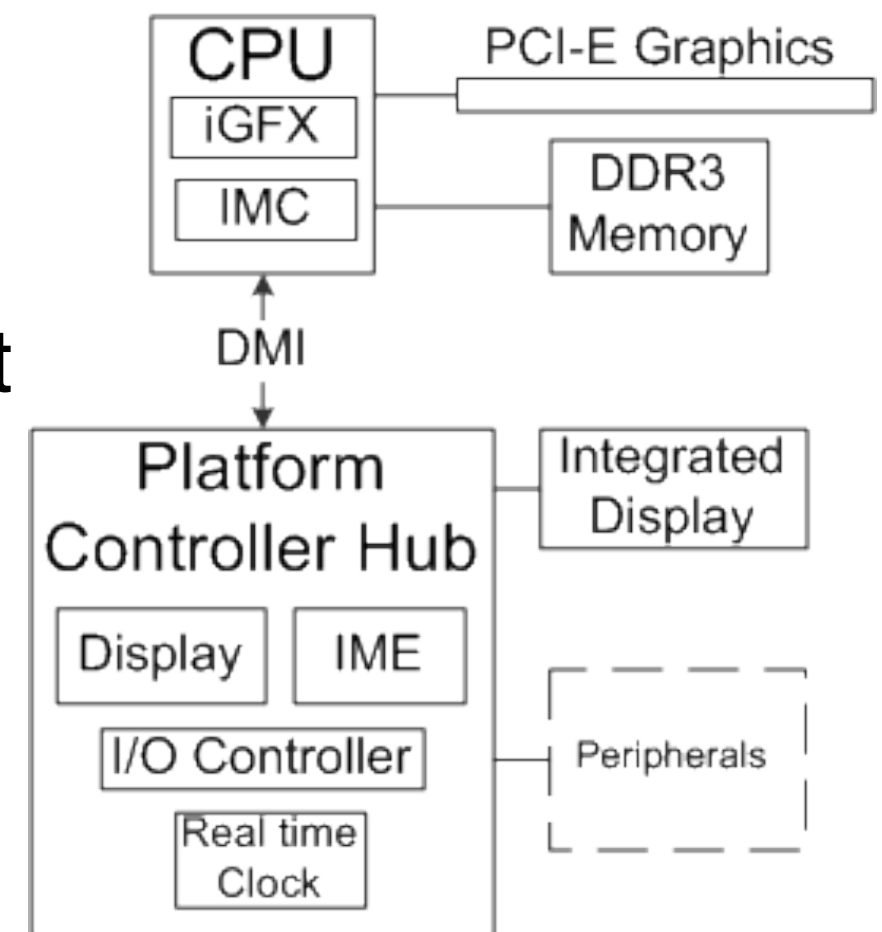
On-die Interconnect (it is not a bus!)

- **High Bandwidth, Low Latency, Modular**
- **Ring** based interconnect between : Cores, Graphics CPU, Last Level Cache (LLC) and System Agent domains
- Composed of 4 rings :
 - ✓ 32 Byte data, Request, Acknowledge & Snoop
 - ✓ Fully pipelined at core frequency/voltage: bandwidth, latency and power scale with cores
- Massive ring wire routing runs over the LLC with no area impact (already paid by LLC)
- Access on ring always picks the shortest path to minimize latency
- Distributed arbitration, sophisticated ring protocol to handle coherency, ordering, and core interface
- Scalable to servers with large number of processors



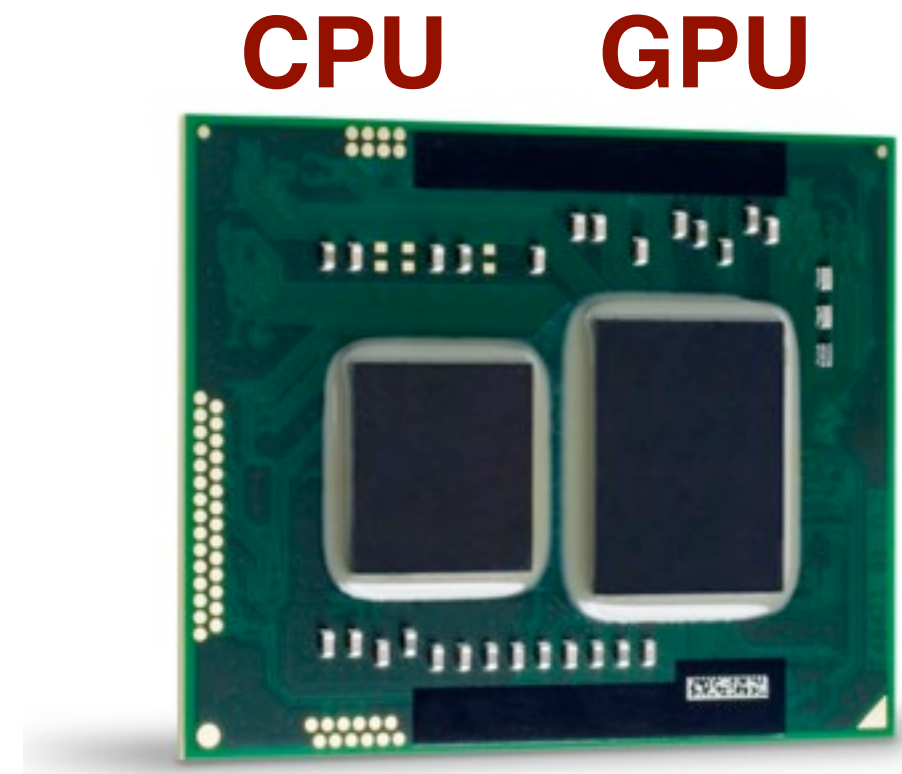
System agent

- An **arbiter** that handles all accesses from the ring domain and from I/O (PCIe* and DMI) and routes the accesses to the right place
- PCIe controllers connect to external PCIe devices
- The PCIe controllers have different configuration possibilities
- DMI controller connects to the Platform Controller Hub (PCH) chipset
- Integrated display engine, Flexible Display Interconnect, and Display Port, for the internal graphic operations
- Memory controller



Graphics Processor

- An embedded GPU
- Started in Nehalem micro-architecture as Multi-Chip Module and heterogeneous IC manufacturing process
- Here embedded into the same IC (SoC) using the same process technology
- Functionality
 - ✓ Multi-media and gaming oriented
 - ✓ HW support for high-performance video encoding/decoding



Versions (branded as Core i3, i5, i7)

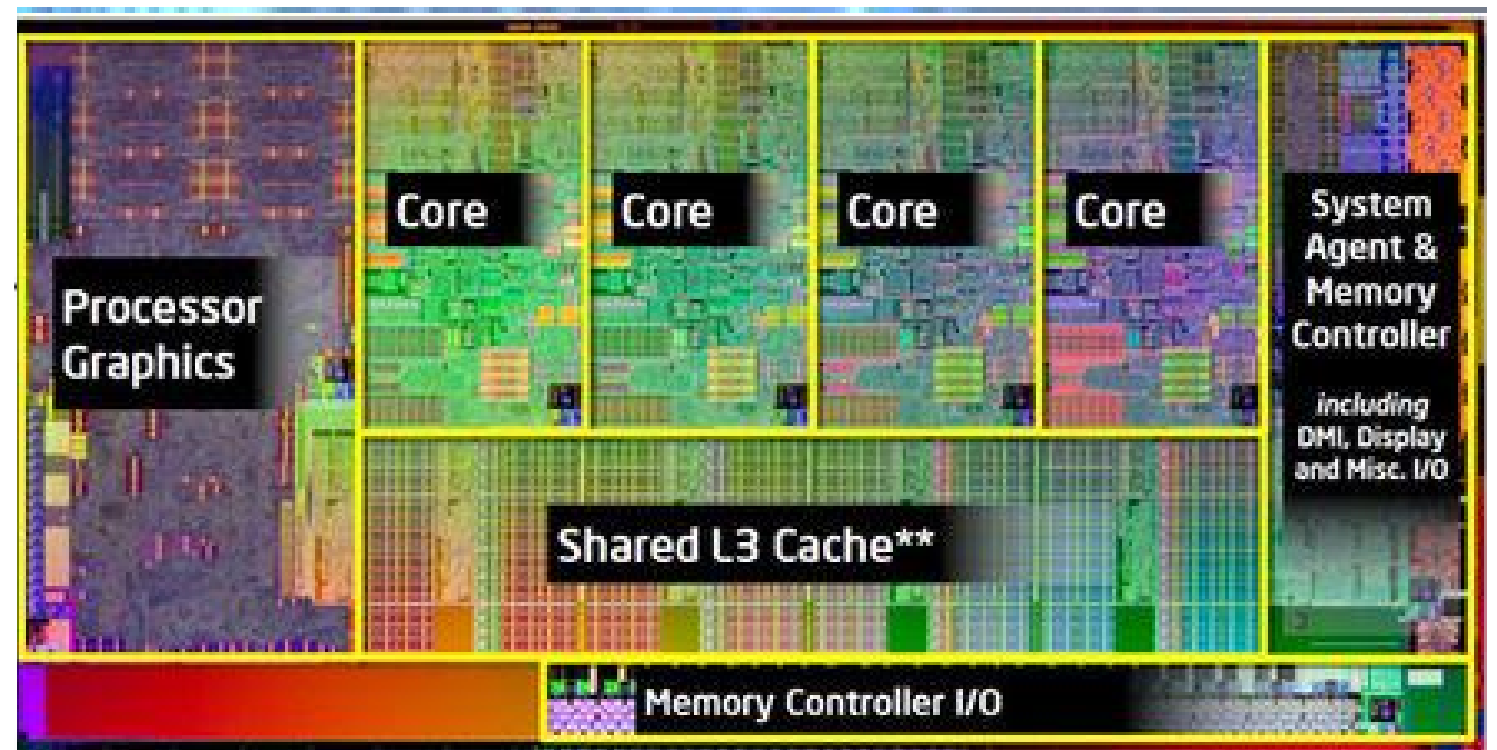
Die Code Name	CPUID	Stepping	Die size	Transistors	Cores	GPU EUs	L3 Cache
Sandy Bridge-HE-4	0206A7h	D2	216 mm ²	1.16 billion	4	12	8 MB
Sandy Bridge-H-2		J1	149 mm ²	624 million	2		4 MB
Sandy Bridge-M-2		Q0	131 mm ²	504 million		6	3 MB
Sandy Bridge-EP-8	0206D6h	C1	435 mm ²	2.27 billion	8	N/A	20 MB
	0206D7h	C2					
Sandy Bridge-EP-4	0206D6h	M0	294 mm ²	1.27 billion	4	N/A	10 MB
	0206D7h	M1					
Ivy Bridge-M-2	0306A9h	P0	94 mm ² ^[11]	1.4 billion ^[14]	2	6 ^[12]	3 MB ^[13]
Ivy Bridge-H-2		L1	118 mm ² ^[11]		2	16	4 MB
Ivy Bridge-HE-4		E1	160 mm ² ^[11]		4	16	8 MB
Ivy Bridge-HM-4		N0	133 mm ² ^[11]		4	6	6 MB ^[13]

Note the impact of the cache memory size on the total die size

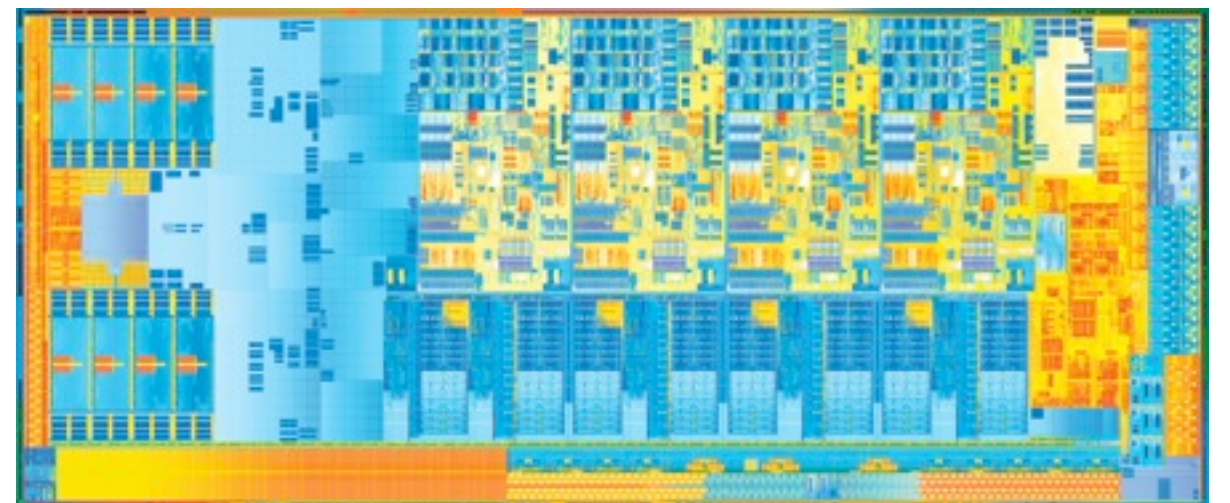
Dies

- For most dense configuration
 - ✓ Up to 2 billion transistors on the single die for Sandy Bridge
 - ✓ 435mm² area
 - ✓ This is huge !

Sandy bridge

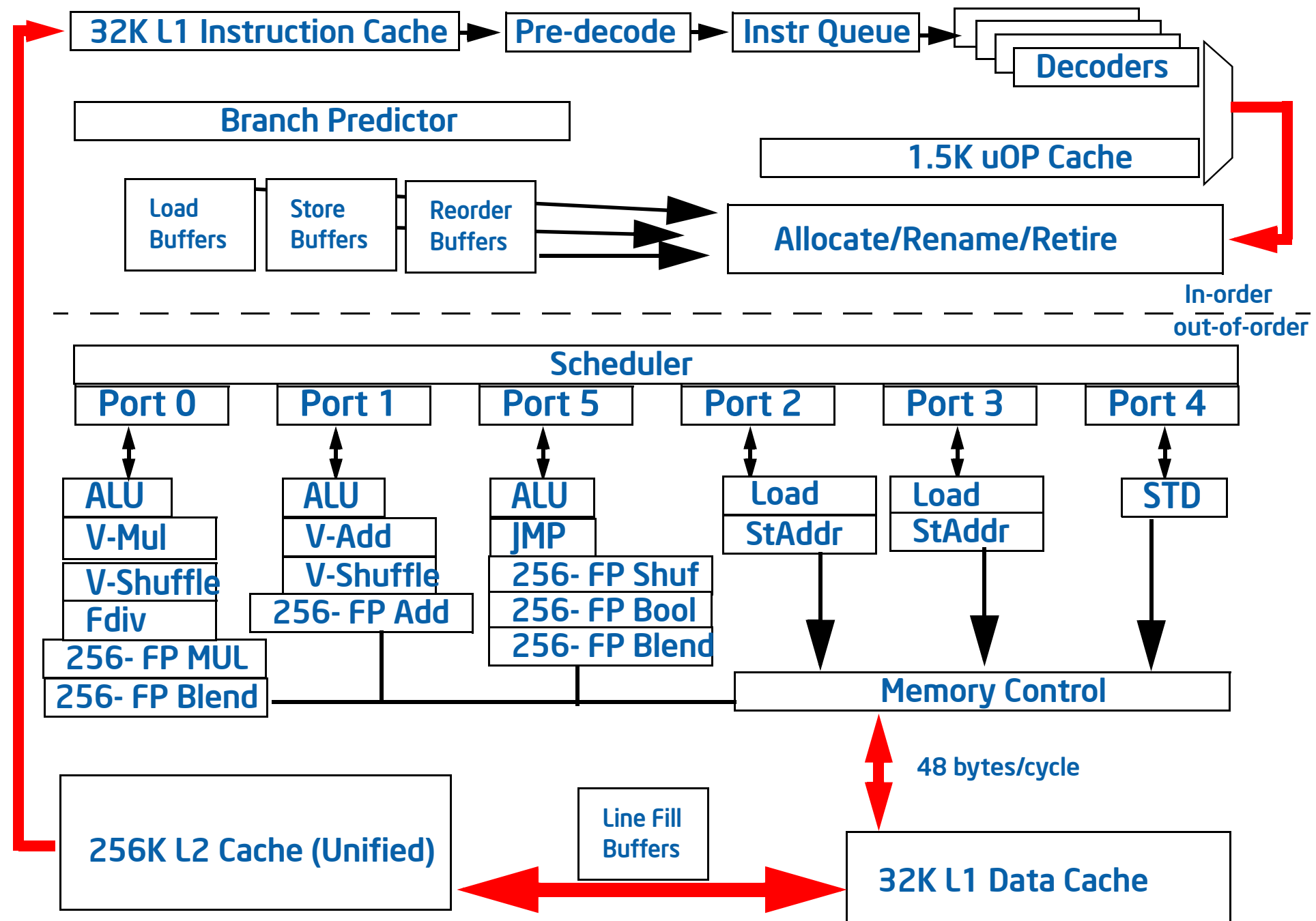


Ivy bridge



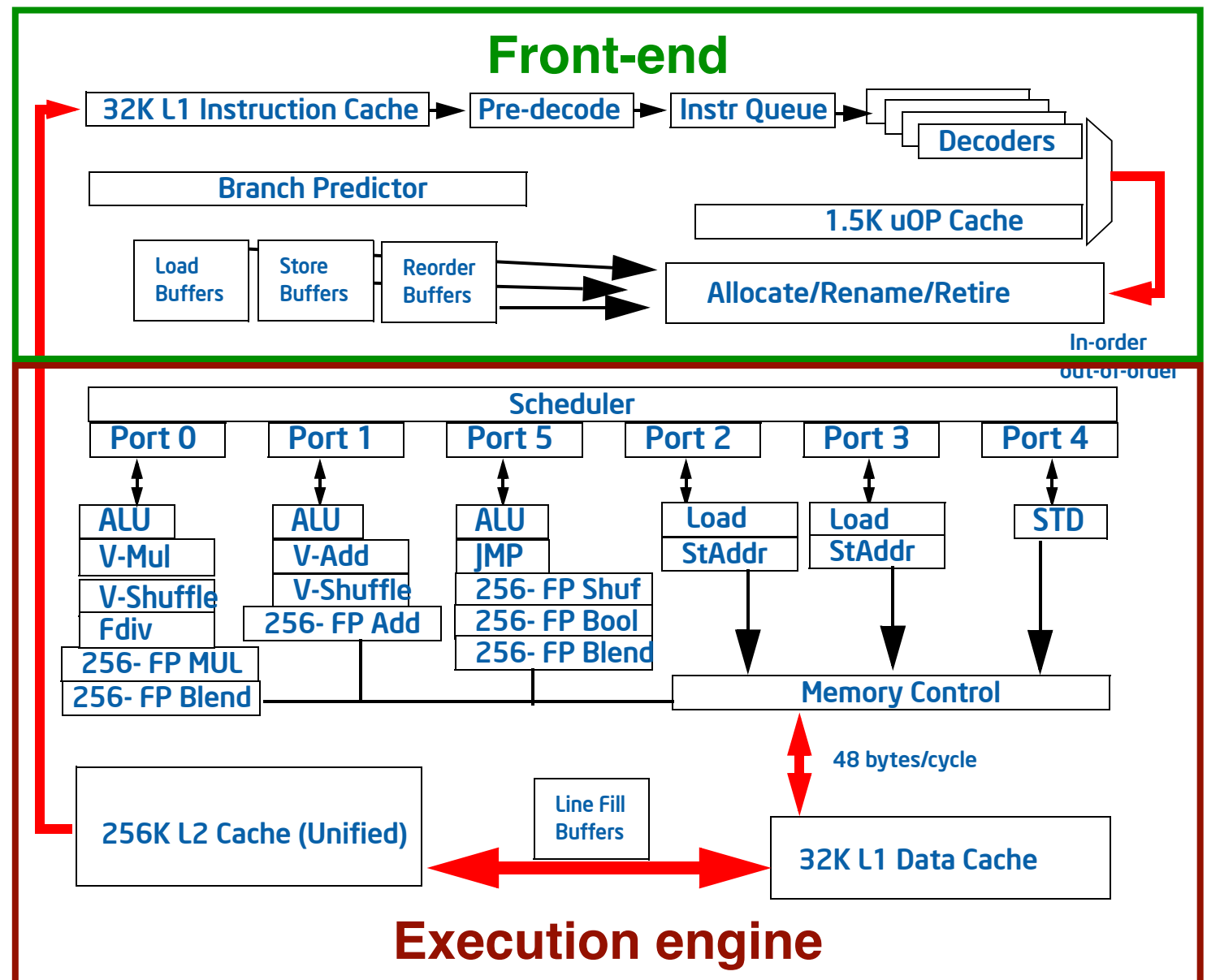
SandyBridge : core architecture

Single core pipeline



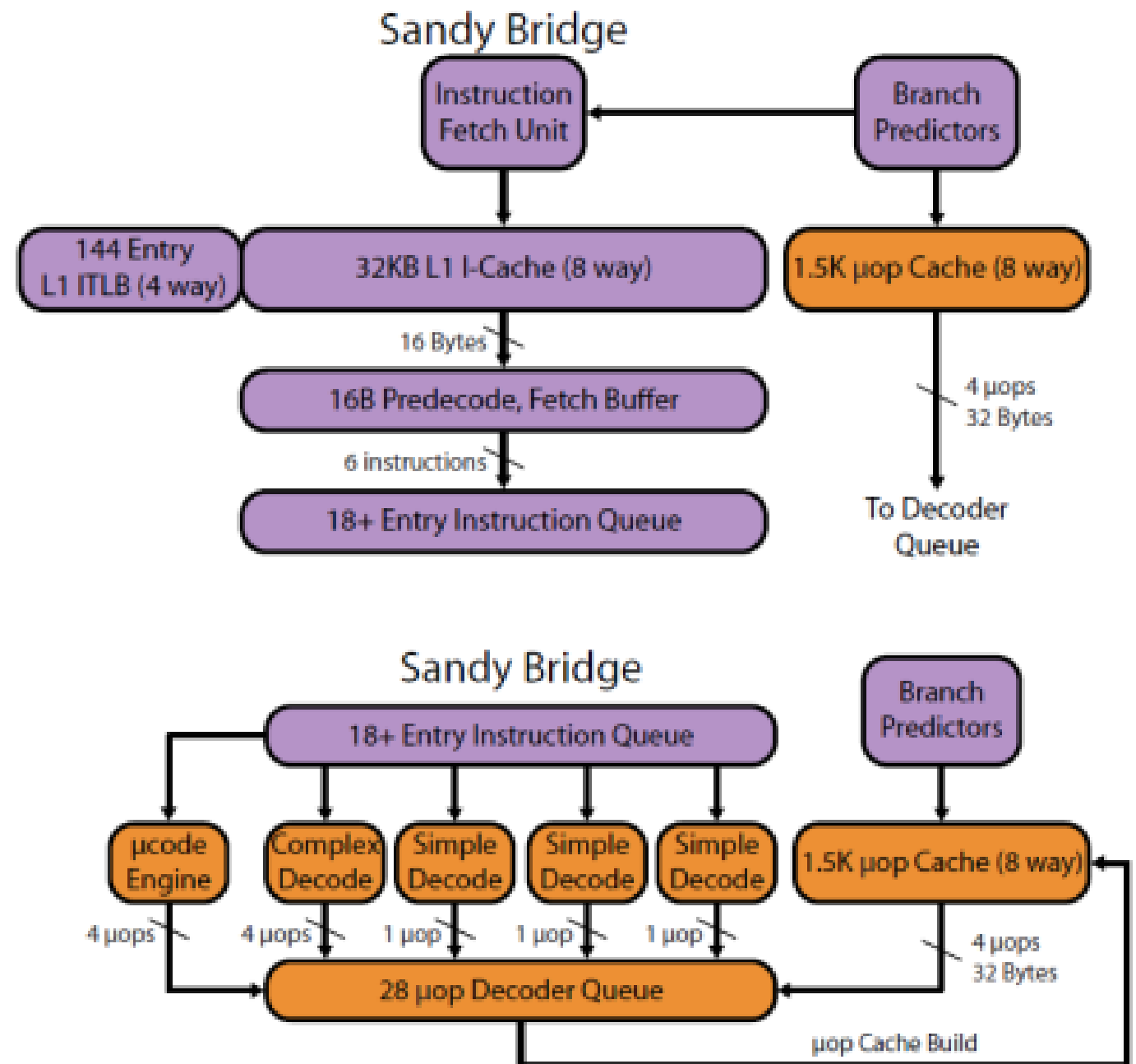
SandyBridge Pipeline overview

- An **in-order issue front-end** that fetches instructions and decodes them into micro-ops (similar to NetBurst)
- The front-end feeds the next pipeline stages with a continuous stream of micro-ops from the most likely path that the program will execute
- An **out-of-order, superscalar execution engine** that dispatches **up to six micro-ops to execution per cycle**
- The allocate/rename block reorders micro-ops to “dataflow” order so they can execute as soon as their sources are ready and execution resources are available



Front-end

- Similar to Net Burst
- Instruction Cache 32KB
8-way + decoded instruction cache (micro-ops)
- Pre-decoding and instruction queuing
- 3 simple decode engines
- 1 complex decode engine



Front-end functionality

- **Legacy DecodePipeline**

- ✓ Decode instructions to micro-ops, delivered to the micro-op queue and the Decoded ICache
- ✓ **Provides the same decode latency and bandwidth as prior Intel processors**

- **Decoded ICache**

- ✓ Provide stream of micro-ops to the micro-op queue

- **MSROM**

- ✓ Complex instruction micro-op, accessible from both Legacy Decode Pipeline and Decoded ICache

- **Branch Prediction Unit (BPU)**

- ✓ Determine next block of code to be executed and drive lookup of Decoded ICache and legacy decode pipelines

- **Micro-op queue**

- ✓ Queues micro-ops from the Decoded ICache and the legacy decode pipeline

Out-of-order engine

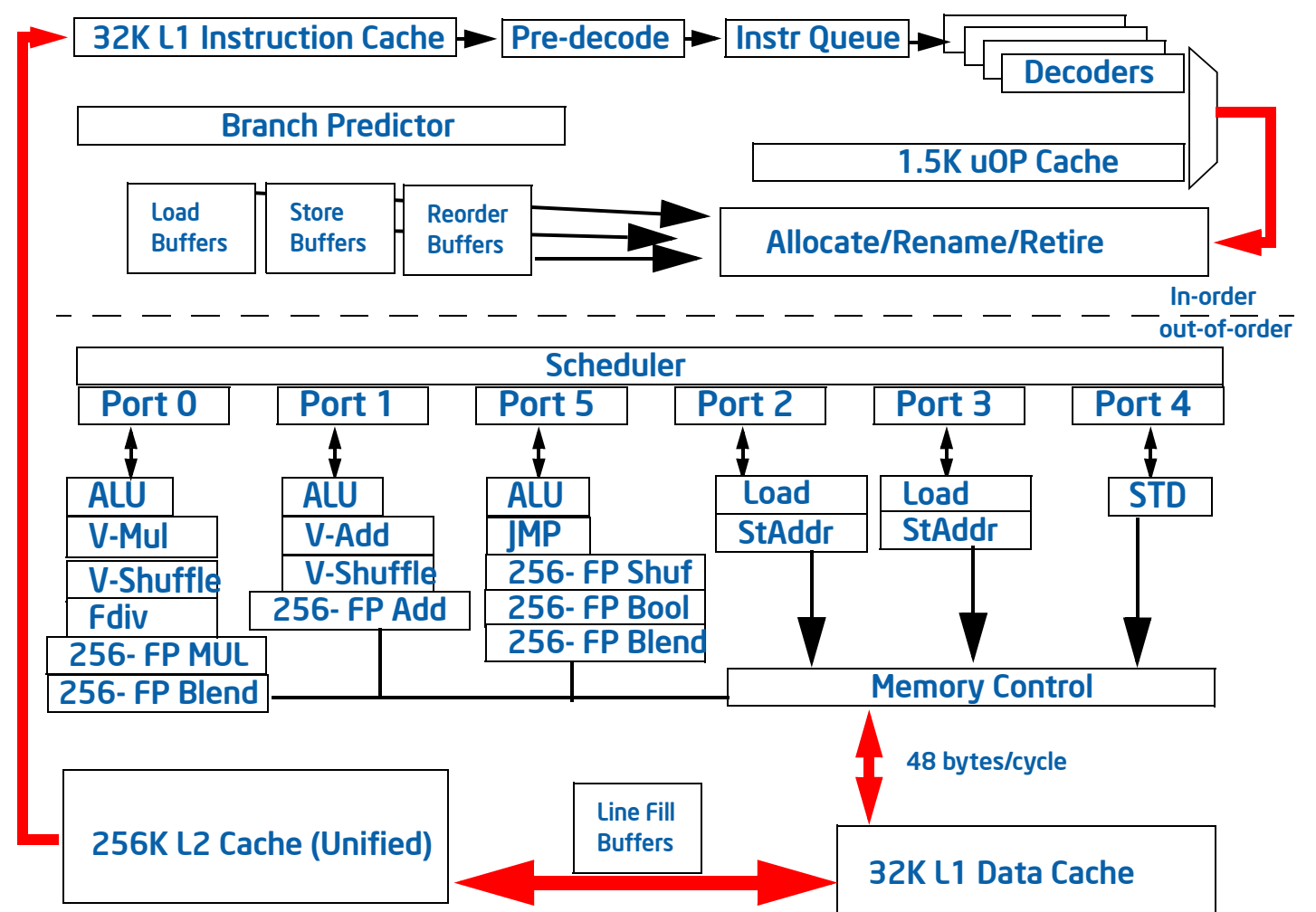
- Improve ILP by detecting dependency chains and by executing them out-of-order while maintaining the correct data flow
- When a dependency chain is waiting for a resource, such as a second-level data cache line, it sends micro-ops from another chain to the execution core
- Designed with power savings in mind too !
- Components :
 - ✓ **Renamer** — moves micro-ops from the front-end to the execution core; eliminates false dependencies among micro-ops, thereby enabling out-of-order execution of micro-ops
 - ✓ **Schedule** — queues micro-ops until all source operands are ready; schedules and dispatches ready micro-ops to the available execution units in as close to a first in first out (FIFO) order as possible
 - ✓ **Retirement** — retires instructions and micro-ops in order and handles faults and exceptions

Execution core

- Superscalar core that processes instructions in out-of-order fashion
- The out-of-order core consist of **three execution stacks**, each stack encapsulates a certain type of data :

- ✓ General purpose integer
- ✓ SIMD integer and X86 floating point
- ✓ X87 floating point instructions

- The execution core also contains connections to and from the cache hierarchy
- The loaded data is fetched from the caches and written back into one of the stacks
- The scheduler can dispatch up to six micro-ops every cycle, one on each port, specialized for certain functionality



Dispatch Port and Execution Stacks

Functional specialisation of execution ports

	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
Integer	ALU, Shift	ALU, Fast LEA, Slow LEA, MUL	Load_Ad dr, Store_ad dr	Load_Ad dr Store_ad dr	Store_dat a	ALU, Shift, Branch, Fast LEA
SSE-Int, AVX-Int, MMX	Mul, Shift, STTNI, Int- Div, 128b-Mov	ALU, Shuf, Blend, 128b-Mov			Store_dat a	ALU, Shuf, Shift, Blend, 128b-Mov
SSE-FP, AVX- FP_low	Mul, Div, Blend, 256b-Mov	Add, CVT			Store_dat a	Shuf, Blend, 256b-Mov
X87, AVX- FP_High	Mul, Div, Blend, 256b-Mov	Add, CVT			Store_dat a	Shuf, Blend, 256b-Mov

Cache hierarchy

- The cache hierarchy contains a first level instruction cache, a first level data cache (**L1** DCache)
- Unified second level (**L2**) cache for each core (shared by instructions and data)
- All cores in a physical processor package are connected to a shared last level cache (**LLC**) via a ring connection

Level	Capacity	Associativity (ways)	Line Size (bytes)	Write Update Policy	Inclusive
L1 Data	32 KB	8	64	Writeback	-
Instruction	32 KB	8	N/A	N/A	-
L2 (Unified)	256 KB	8	64	Writeback	No
Third Level (LLC)	Varies, query CUID leaf 4	Varies with cache size	64	Writeback	Yes

Cache hierarchy

- L1 is private to a core
- L1D cache may be shared by two logical processors if the processor support HyperThreading
- The L2 cache is shared by instructions and data
- The caches use the services of the
 - ✓ Instruction Translation Lookaside Buffer (ITLB),
 - ✓ Data Translation Lookaside Buffer (DTLB) and
 - ✓ Shared Translation Lookaside Buffer (STLB)
- These are used to translate linear addresses to physical address
- Data coherency in all cache levels is maintained using the MESI protocol

Load operation latency

Level	Latency (cycles)	Bandwidth (per core per cycle)
L1 Data	4 ¹	2 x16 bytes
L2 (Unified)	12	1 x 32 bytes
Third Level (LLC)	26-31 ²	1 x 32 bytes
L2 and L1 DCache in other cores if applicable	43- clean hit; 60 - dirty hit	

- When an instruction reads data from a memory location that has write-back (WB) type, the processor looks for it in the caches and memory
- This is best case latency, the actual latency will vary depending on the cache queue occupancy, LLC ring occupancy, memory components
- Each cache line in the LLC holds an indication of the cores that may have this line in their L2 and L1 caches → if this is the case lookup
 - ✓ The lookup is called "**clean**" if it does not require fetching data from the other core caches
 - ✓ The lookup is called "**dirty**" if modified data has to be fetched from the other core caches and transferred to the loading core

Store

- When an instruction writes data to a memory that has a write back memory type, the processor first ensures that it has the line containing this memory location in its L1 DCache
- If the cache line is not there (and in the right coherency state), the processor fetches it from the next levels of the memory hierarchy using a Read for Ownership request in the specified order:
 - ✓ L1 DCache
 - ✓ L2
 - ✓ Last Level Cache
 - ✓ L2 and L1 DCache in other cores, if applicable
 - ✓ Memory
- Once the cache line is in the L1 DCache, the new data is written to it, and the line is marked as Modified state
- Low latency cost, except if multiple consecutive write cache misses

SIMD

- Legacy : from MMX to SSE4.2 + **AVX**
- **AVX**
 - ✓ Support for **256-bit wide vectors** and 16 SIMD register set
 - ✓ **256-bit floating-point instruction set** enhancement with up to 2X performance gain relative to 128-bit Streaming SIMD extensions.
 - ✓ Instruction syntax support for generalized **three-operand syntax** to improve instruction programming flexibility and efficient encoding of new instruction extensions
 - ✓ **Enhancement of legacy** 128-bit SIMD instruction extensions to support three-operand syntax and to simplify compiler vectorization of high-level language expressions.
 - ✓ Support flexible deployment of 256-bit AVX code, 128-bit AVX code, legacy 128-bit code and scalar code

Power and thermal management

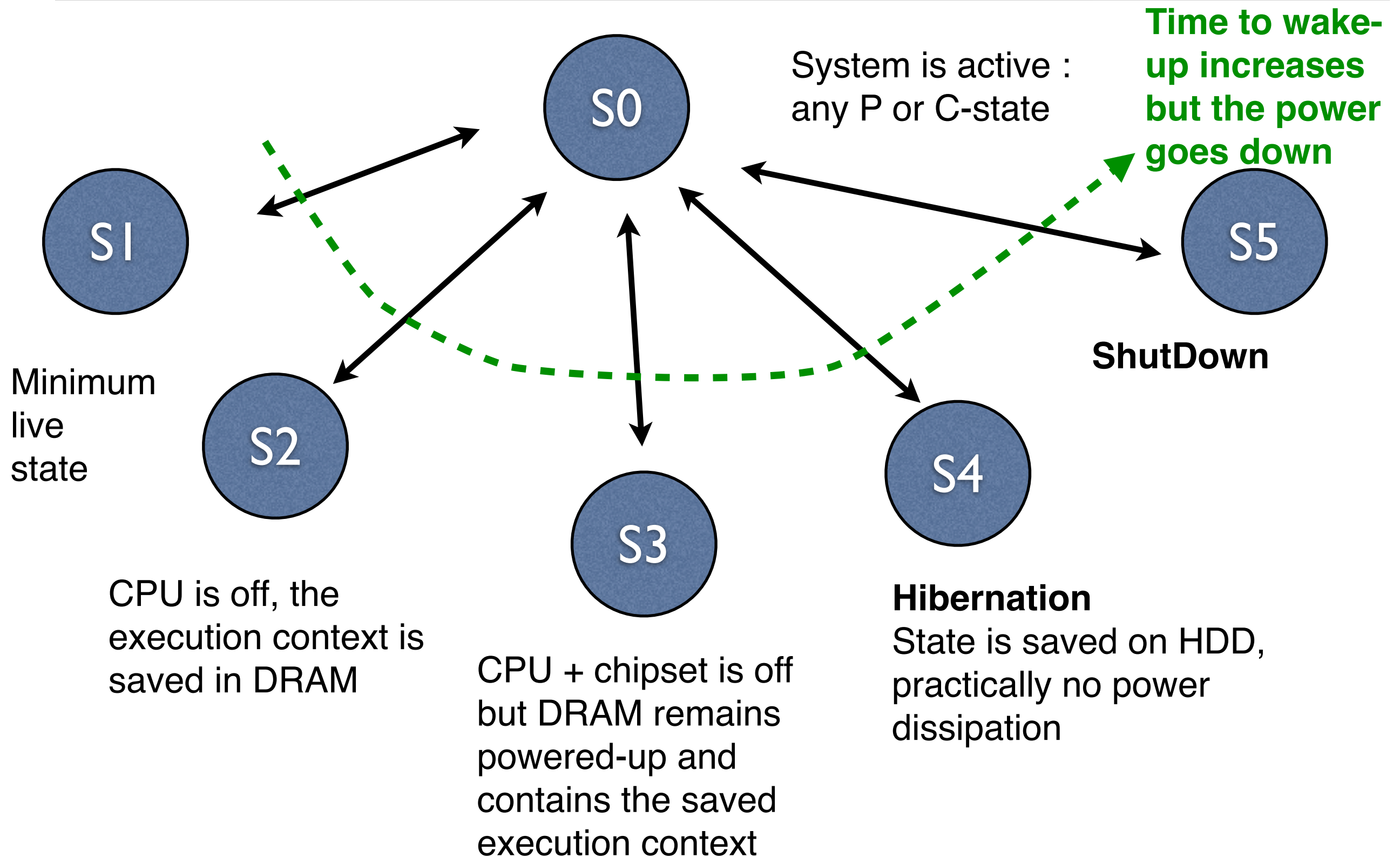
Power in ICs

- Power is a problem in current ICs : limited frequency speed-ups from node to node because of the power and associated thermal
- Has two components
 - ✓ **Dynamic** — when gates are toggling (output of the gates changes due to input changes)
 - ✓ **Static** — due to leakage, transistors are not perfect switches
- While in the past dynamic component has been pre-dominant, over the years static power dissipation became as important
- To deal with static power dissipation we need to “cut” the
 - ✓ clock supply — **clock gating** or reduce clock period
 - ✓ V_{dd} — **power gating**
- Most of the ICs implement these techniques; for general purpose CPU it is important to have an interface with the layers above

Hierarchical management

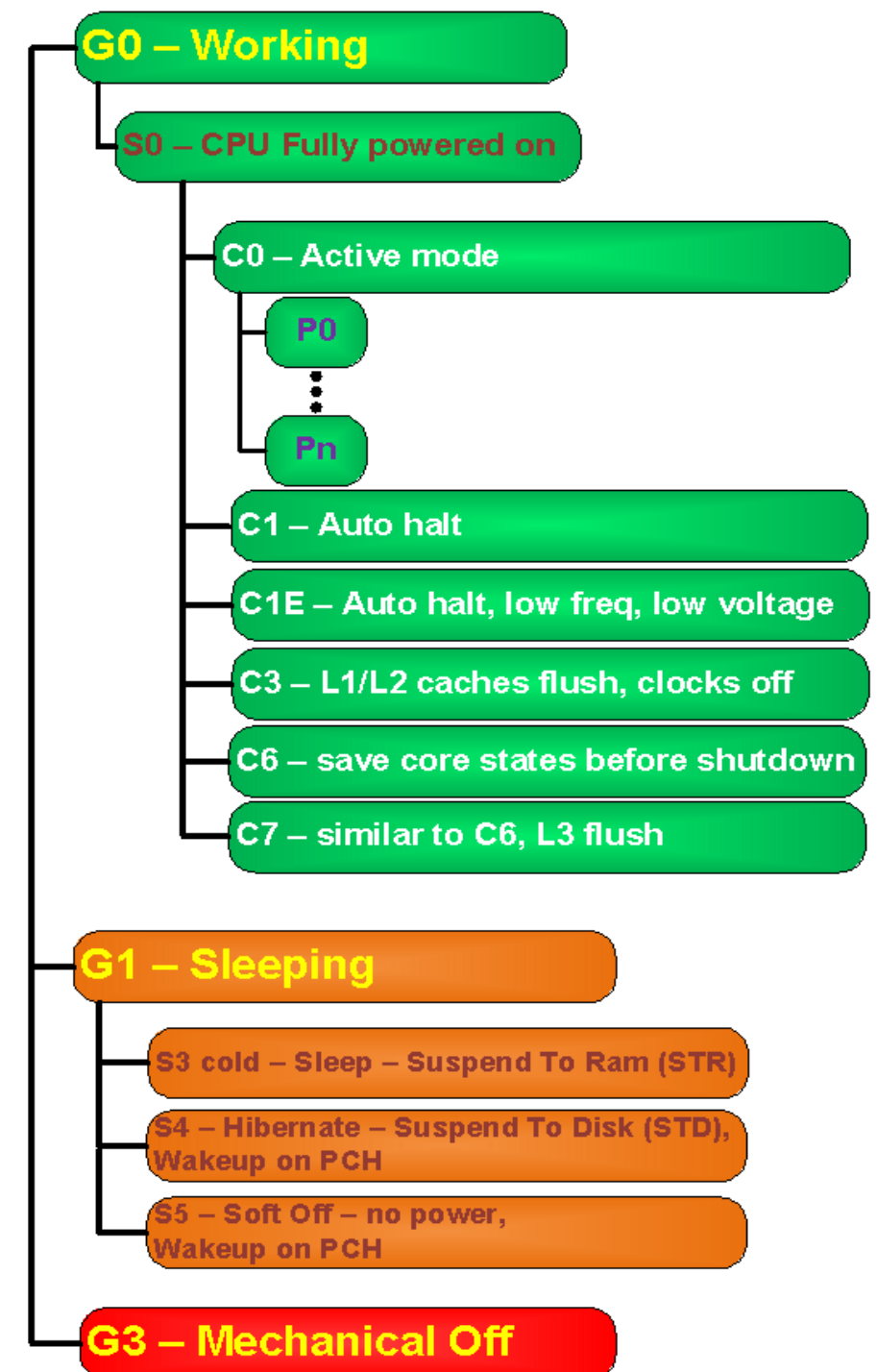
- Previously done in BIOS → moves from firmware to OS using dedicated core (here a core 1M transistors! ~ 486)
- **Advanced Configuration and Power Interface** (ACAPI)
 - ✓ Open standard for platform independent hardware discovery, configuration, power management and monitoring (Intel, HP, etc.)
 - ✓ Operating System-directed configuration and Power Management (OSPM)
- Different CPU state classes :
 - ✓ **S-State — System Sleep States**
 - system level sleep state
 - ❖ **C-State — Microprocessor and Package Idle States**
 - off, can wake up (4 different states)
 - ➔ **P-State — Microprocessor Performance State**
 - CPU in active state defines 6 different states (DVFS couples)
 - ✓ **T-State — Microprocessor Throttle States**
 - we know little about these ...

S-States — System Sleep States



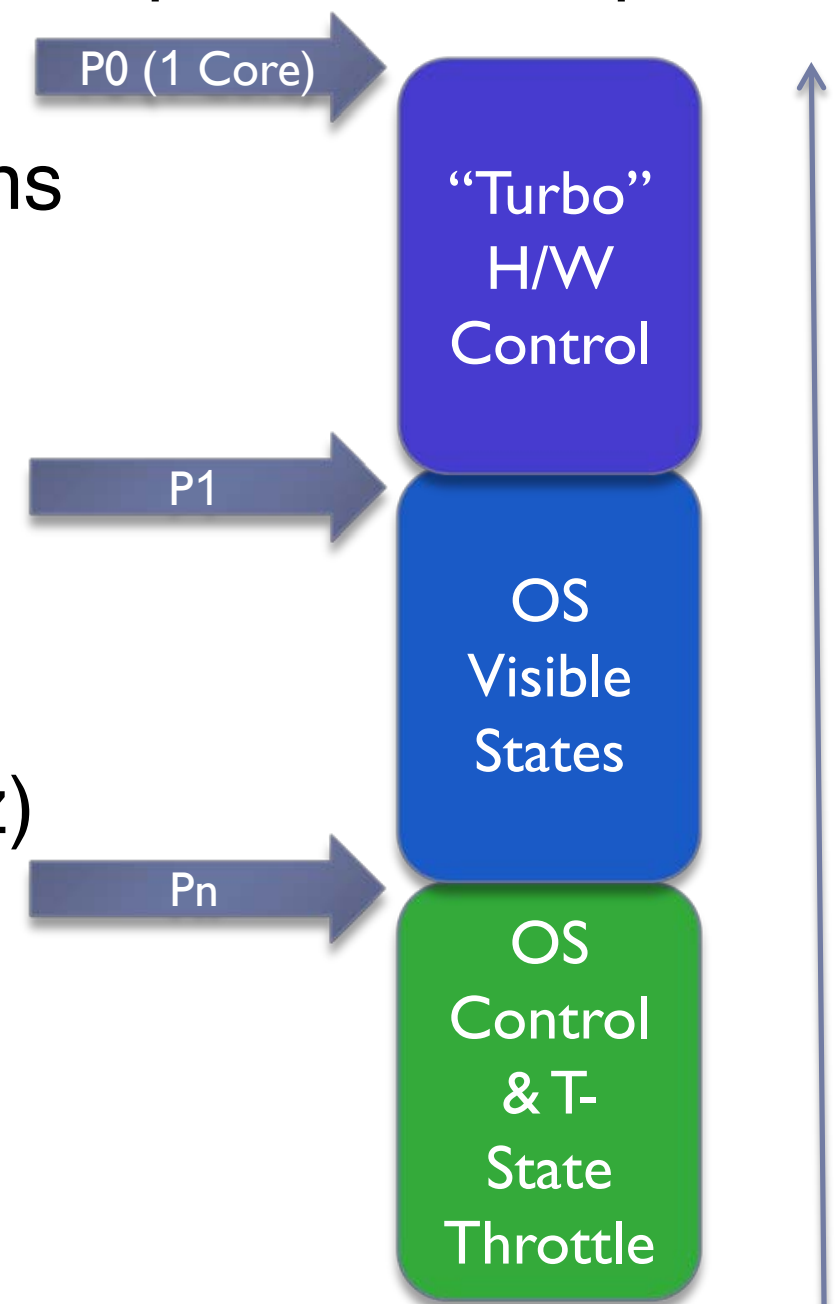
C-States

- When fully powered (CPU is in S0 state) it can be in one of the C-States:
 - ✓ C0 — cores are in one of the P-States (that is having one of the Vdd, Clk couple)
 - ❖ Defines P sub-states
 - ✓ C1 — no execution, all Clk signals are off (Clk gating)
 - ✓ C3 — caches are empty but Vdd is applied, core is active
 - ✓ C6 — no Vdd @ core level to avoid leakage → system state is saved in LLC practically 0W, core is inactive



P-States

- **Thermal Design Power (TDP)** — maximum amount of power the cooling system in a computer is required to dissipate
- Processor operates at power levels higher than rated TDP limit for short durations
 - ✓ P0 is max possible frequency
 - ✓ P1 is guaranteed frequency
 - ✓ Pn is the energy efficient state
- In between OS controls Pn-P1 range
- P1-P0 has significant frequency range (GHz)
 - ✓ P1 to P0 range is fully H/W controlled
 - ✓ User preferences and policies
 - ✓ Single thread or lightly loaded applications
- GFX $\leftrightarrow$ CPU balancing



P-states examples

- P-States implement the idea of **Dynamic Voltage and Frequency Scaling (DVFS)**
- Each P-State defines a couple : Vdd, F
(Do you remember the link between the Vdd and F ?)
- P-States and corresponding DVFS couples
 - ✓ Example for a mobile CPU :

P-State	Frequency	Voltage	Power
P0	1.6 GHz	1.484 V	25 Watts
P1	1.4 GHz	1.420 V	~17 Watts
P2	1.2 GHz	1.276 V	~13 Watts
P3	1.0 GHz	1.164 V	~10 Watts
P4	800 MHz	1.036 V	~8 Watts
P5	600 MHz	0.956 V	6 Watts

Intel TurboBoost Technology

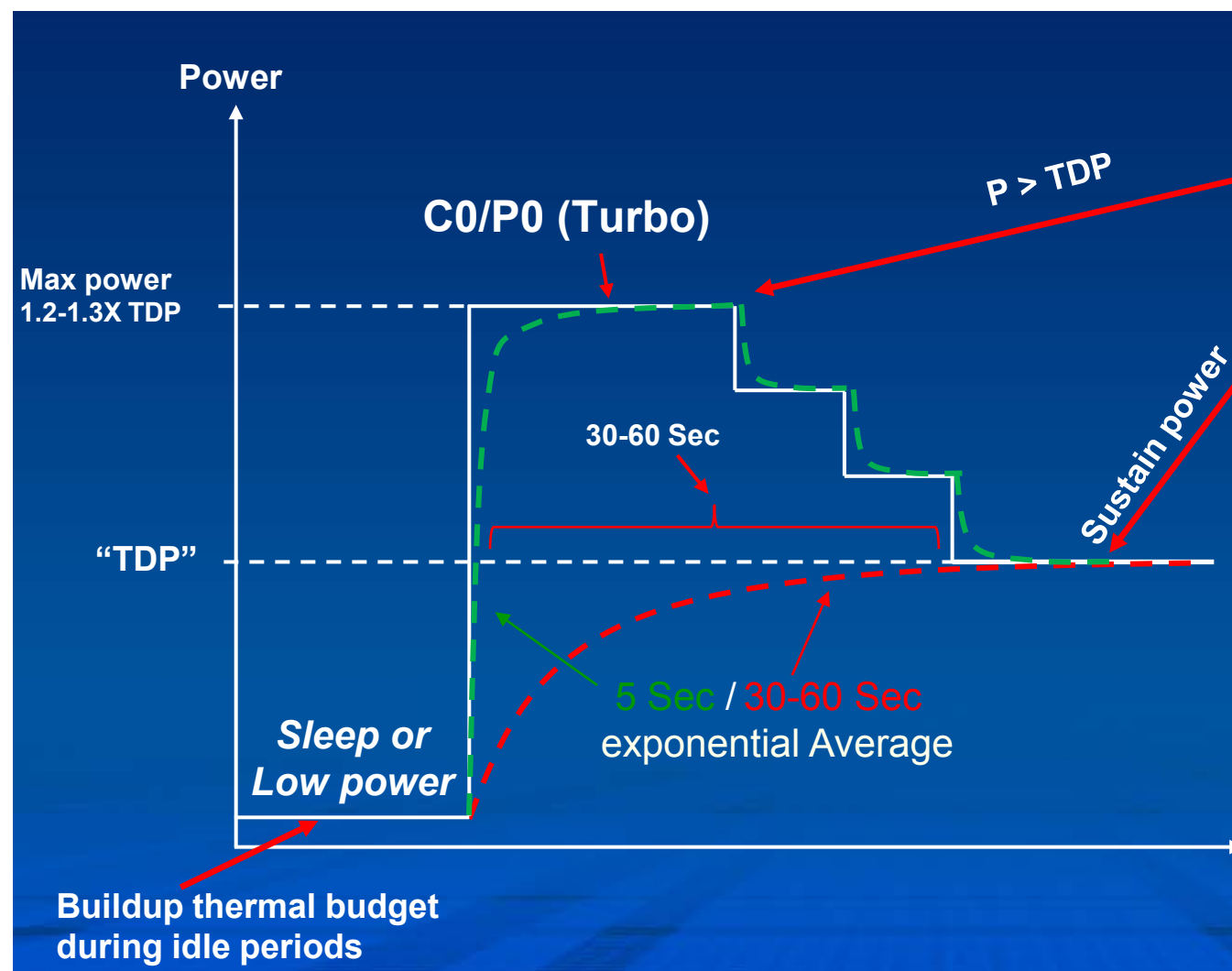
- **Performance on Demand**

- ✓ CPU can run faster than base operating frequency
- Working within power, current and temperature constraints
- Dependent on number of active cores
- Highest frequency is with 1 core (gets all the headroom)
- Example: base $F=2.5\text{GHz}$
 - more F if less cores running @ the same time

N° de cores	N° of Turbo Steps	Max F [GHz]
3 or 4	7	$3.2 = 2.5 + 7 * 100$
2	9	$3.4 = 2.5 + 9 * 100$
1	10	$3.5 = 2.5 + 10 * 100$

How we can do TurboBoost ?

- After idle periods, the system accumulates “energy budget” and can accommodate high power/performance for up to a minute
- In Steady State conditions the power stabilises on TDP, possibly at higher then nominal frequency



Thermal management

- On die thermal sensors
 - ✓ 12 sensors on each CPU core + Graphics processor, ring and System Agent
 - ✓ Operating range 50-100°C
- Temperature reporting
 - ✓ Maximum reading of each functional block
 - ✓ Maximum reading of the total chip
- Used for:
 - ✓ Critical thermal protection
 - ❖ Notification, throttle and shutdown
 - ❖ Programmable throttle temperature
 - ✓ Leakage calculation of power meter
 - ✓ Power optimization algorithms
 - ✓ External system controls (e.g. Fan control)

